JOURNAL OF COMPUTER AND SYSTEM SCIENCES 47, 77–120 (1993)

A Logic for Programming with Complex Objects*

MICHAEL KIFER AND JAMES WU†

Department of Computer Science, SUNY at Stony Brook, Stony Brook, New York 11794

We present a logic for reasoning with complex objects, which is a repaired and significantly extended version of Maier's O-logic [43]. The logic naturally supports complex objects, object identity, and deduction, and has several other interesting features. It elegantly combines object-oriented and value-oriented paradigms and, in particular, contains all of predicate calculus as a special case. The revised O-logic has a sound and complete resolution-based proof procedure.

1. PREFACE

In the past few years, considerable interest arose in the so-called object-oriented approach to databases. Although there is no consensus regarding what an object-oriented approach precisely means, either in programming languages or in databases, a number of concepts have been identified as its most salient features. According to [10, 51, 58, 59] and a number of other surveys, these concepts are: complex objects, inheritance, class/subclass classification, and object identity. In parallel, the deductive approach gained enormous popularity both in programming languages and databases, and it became apparent that deduction is very desirable in object-oriented languages as well. Furthermore, the desire here is to have a "logic-programmable" deduction. In this respect, relational calculus-like deduction does not fully meet this goal, and a full-fledged locic is needed. Many attempts have been made to combine the two approaches [1, 2, 4, 8, 9, 13, 14, 38, 43, 45]. Although each approach was an important step forward; in our opinion, none of them succeed in achieving the above goals satisfactorily.

In [52], it is claimed that the deductive approach is intrinsically "value-oriented" and therefore cannot be combined with the inherently object-oriented features, such as object identity. The work reported here seems to suggest otherwise: we present a logic language with a well-defined semantics that extends predicate calculus and accommodates deduction with complex objects and object identity. Furthermore, value-oriented entities (i.e., the ones that are determined solely by the values of

* Supported in part by the NSF Grant DCR-8603676; a preliminary report on this research appeared in "Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems."

† Present address: Renaissance Software, 175 S. San Antonio Rd., Los Altos, CA 94022.

0022-0000/93 $5.00

their attributes) and object-oriented entities (i.e., those determined by object identities) can be mixed freely in one program and even in one clause.

Our work is based on O-logic proposed by Maier [43], which, unfortunately, had a number of semantic flaws. Luckily, Maier's basic insight into the problem proved to be very fruitful, and our revised version of O-logic does not suffer from these difficulties. We have also significantly extended the semantics to include sets, inconsistency-tolerance, and equality and provided a sound and complete resolution-based proof procedure, which makes O-logic also computationally attractive.

Table I illustrates the standing of various proposals with respect to the goals declared earlier. Because of the abundance of different logic-based approaches to object-oriented databases, we are unable to discuss all of them in detail; the table indicates only the most salient features of each proposal. A more thorough comparison with some of these works appears in the main body of the paper. It should be noted that none of the approaches listed below fully incorporates the inheritance mechanism. What is referred to as "inheritance" in most cases is a simple class–subclass classification, but subclasses do not really inherit values from superclasses. An extension of the present work that accommodates inheritance with overriding is presented in [34]. In Table I, (*a*) indicates calculus-based languages where recursion is not allowed. Although recursion can be expressed using the power-set operator [1, 26], calculus-based languages are too inflexible for general-purpose logic programming. Superscipt (*b*) indicates proposals whose deduction is restricted to a subset of logic (normally Horn or extended Horn rules) or those relying on extra-logical features. (*c*) marks proposals that restrict sets to be finite. Furthermore, some works either permit no sets or allow only acyclic objects; this is also indicated in the table.

There is also a different (orthogonal, in a sense) stream of object-oriented database research that does not state the development of an object-oriented logic

TABLE I

Language	Complex objects	Class/subclass	Object identity	Deduction
NFNF [45]	Acyclic	No	No	Yes[a]
LDM [38]	Yes	No	Yes	Yes[a]
CCO [9]	Acyclic	Possible	No	Yes[b]
LDL [13]	Acyclic	No	No	Yes[b]
COL [2]	Acyclic[c]	No	No	Yes[b]
ELPS [39]	Acyclic[c]	No	No	Yes
Abiteboul–Beeri [1]	Acyclic	No	No	Yes[a]
Rubinski [46]	Yes	Yes	Yes	Yes[a]
Ψ-terms [14]	Yes	Yes	No	Yes[b]
IQL [4]	Yes	Yes	Yes	Yes[b]
C-Logic [19][1]	Sets only	Yes	Yes	Yes
O-logic (Maier's)	No sets	Yes	Yes	Ill-defined
O-logic (ours)	Yes	Yes	Yes	Yes

[1] C-logic [19] is a proper subset of the extended O-logic presented in this paper.

language as its main goal (e.g., [11, 36, 40, 41]). The effort here is directed towards designing a *procedural* object-oriented language supported by a formal data model. In the present paper, we will not discuss and/or compare these works with ours.

This manuscript mostly deals with general aspects of the proposed logic, such as its model theory, resolution-based theorem proving and treatment of sets. However, Section 10 also outlines a logic programming semantics, thereby laying foundations for a theory of object-oriented logic programming.

The relationship of our work to frame-based languages in AI is worth noting also. Marriage of frames with deduction has been a hit in AI for quite some time (e.g., [23]), but no semantically solid and yet comprehensive theory has ever been proposed. Frames are essentially scaled-down versions of complex objects (plus inheritance), and pointers referring to other frames are conceptually nothing else but object identities. Although Hayes has shown that certain features attributed to frames can be translated into predicate calculus [27], this translation does not provide a direct semantics for frames and does not capture many aspects naturally enough. To our knowledge, the present work is the first comprehensive approach that directly marries deduction with several important aspects of frames with the blessing of a respectable logical theory; [32, 34] are further steps in this direction.

2. Introduction to O-logic

From now on, the term O-logic will refer to our version of the formalism; when mentioning Maier's original O-logic [43], suitable qualifiers will be added. In this section we introduce the syntax and semantics of our language. Basic knowledge of logic programming and deductive databases on the level of the first few chapters of [42, 53] is assumed.

2.1. *Syntax*

An alphabet of O-logic consists of:

(1) A possibly infinite set **F** of *object constructors*;

(2) A possibly infinite set $\mathbf{A}_{\rightarrow}$ of *single-valued* (also called *functional*) *attributes*;

(3) A possibly infinite set $\mathbf{A}_{\twoheadrightarrow}$ of *set-valued attributes*;

(4) A possibly infinite collection Σ of *class names*;

(5) An infinite set **V** of *object variables*;

(6) Logical connectives $\vee$, $\wedge$, $\Leftarrow$, $\neg$, and quantifiers $\forall$, $\exists$.

Additional alphabet symbols such as parentheses and arrows will be introduced as we go. We assume that the sets **F**, $\mathbf{A}_{\rightarrow}$, $\mathbf{A}_{\twoheadrightarrow}$, **V**, and Σ are disjoint.

Object constructors play the role of *function symbols* in O-logic and we will use the terms "object constructor" and "function symbol" interchangeably. Every

symbol $f \in \mathbf{F}$ has an *arity*—a nonnegative integer specifying the number of arguments f can take. 0-ary symbols in $\mathbf{F}$ are called *constants*.

An *id-term* is a term composed out of object constructors (members of $\mathbf{F}$) and object variables (members of $\mathbf{V}$) in the usual way (e.g., $f(a, g(X, b))$, where f, g are binary object constructors, a and b are constants, and X is a variable). The set of all *ground* (i.e., variable-free) terms is denoted by $\mathbf{F}^*$. Conceptually, each element of $\mathbf{F}^*$ can be viewed as an object *denotation* or "name," which is a logical abstraction corresponding to the concept of *object identity* [29]—an *ad hoc* notion that is omnipresent in object-oriented systems. However, unlike most such languages, we do allow the same object to have more than one denotation (and thus more than one object id). For instance, in O-logic one can state *john* = *father*(*mary*), thereby making *john* and *father*(*mary*) refer to the same object. This view of object ids as object denotations or names seems to be quite common in AI. In contrast, database languages usually insist on uniqueness of an id per object, which stems from the narrow implementational concerns, but is not well-motivated otherwise.

Thoughout this paper we will use symbols starting with lower-case letters to denote ground (i.e., variable-free) terms; symbols beginning with capital letters will be used to denote terms that may or may not contain variables. Emboldened lower-case letters will denote classes (the elements of Σ). Also, to eliminate needless repetitions, we will be consistently using the symbols $\mathbf{V}$, $\mathbf{F}$, $\mathbf{F}^*$, $\mathbf{A}_{\rightarrow}$, $\mathbf{A}_{\twoheadrightarrow}$, Σ to denote the sets of variables, function symbols, etc., of some language $\mathbf{L}$ that will be either known from the context or immaterial.

We define O-terms along the lines of [43], extending them to include sets and class-hierarchies. An *O-term* (an *O-logic term*) is either

(1) A *typing O-term* T: $\mathbf{p}$, where T is an id-term and $\mathbf{p} \in \Sigma$ is a class name;

(2) An *is-a O-term* $\mathbf{p}$: $\mathbf{q}$, where $\mathbf{p}, \mathbf{q} \in \Sigma$ are class names; or

(3) A *complex O-term* $T[funA_1 \rightarrow T_1, \ldots, funA_n \rightarrow T_n, setA_1 \twoheadrightarrow \{S_{1,1}, \ldots, S_{1,m_1}\}, \ldots, setA_k \twoheadrightarrow \{S_{k,1}, \ldots, S_{k,m_k}\}]$, where T is an id-term, $funA_i \in \mathbf{A}_{\rightarrow}$ are functional attributes, and $setA_j \in \mathbf{A}_{\twoheadrightarrow}$ are set-valued attributes; T_i, $S_{j,l}$ are complex O-terms. We will often refer to T as the *id* of the above O-term.

Typing O-terms classify objects into classes (e.g., *john*: **student**), while is-a O-terms organize classes into ISA-hierarchies (e.g., **faculty**: **employee**). Complex O-terms are assertions about various properties of objects represented by id-terms, e.g., *father*(*john*)[*age* → 30, *salary* → 20*K*, *children* ↠ {*mary*, *john*}]. As their name suggests, functional attributes (such as *age* above) are used to represent functional properties of objects (*father*(*john*), in this case), that is, properties that are described by exactly one value-object (30, in our example). The term "value-object" is used informally here and is intended to foster the reader's intuition that, say, 30 is an object (actually, an object denotation) returned by the attribute *age* in the context of the object *father*(*john*). Likewise, set-valued attributes represent properties that return sets of objects as their values (cf. *children*).

To endow the reader with a proper intuition about the intended meaning of

complex O-terms, we will jump ahead and remark that more than one such O-term may describe the same object. For instance, an object *bob* may be described by a pair of complex O-terms $bob[name \rightarrow \text{"}Bob\text{,"}\ children \twoheadrightarrow \{mary\}]$ and $bob[age \rightarrow 50,\ children \twoheadrightarrow \{bill\}]$, each supplying part of the information about *bob*. According to the semantics given in the next subsection, this pair of O-terms is equivalent to the following single O-term: $bob[name \rightarrow \text{"}Bob\text{,"}\ age \rightarrow 50,\ children \twoheadrightarrow \{mary, bill\}]$. Another important observation concerning the definition of complex O-terms is that they may occur inside other O-terms in a "value"-position,[1] e.g., $bob[children \twoheadrightarrow \{mary[age \rightarrow 15], bill\},\ age \rightarrow 50[type \rightarrow \text{"}integer\text{"}]]$. In Section 2.2, we shall see that this is actually equivalent to a conjunction of simpler O-terms: $bob[children \twoheadrightarrow \{mary, bill\},\ age \rightarrow 50]$, $mary[age \rightarrow 15]$, and $50[type \rightarrow \text{"integer"}]$.

An O-term is also a *molecular*[2] *O-formula*. O-formulae are obtained from other (simpler) O-formulae by means of logical connectives $\vee$, $\wedge$, $\neg$, and quantifiers $\exists$ and $\forall$. In the programming examples, however, we will use "&" to represent conjunction of subgoals in rule bodies. Also, as usual, we define the implication $(\phi \Leftarrow \psi)$ to be equivalent to $\phi \vee \neg\psi$.

If all attributes in a complex O-term S of the form $T[\ ]$ are omitted, we can identify S with its object id, the id-term T. This allows us to view id-terms as a special case of complex O-terms. However, to avoid syntactic confusion with predicates (introduced later), we will allow to drop "[ ]" only in O-terms that appear in a "value"-position in another complex O-term (e.g., T_i and $S_{j,k}$ in (3) above). We will thus write $bob[age \rightarrow 30,\ father \rightarrow bill]$ instead of $bob[age \rightarrow 30[\ ],\ father \rightarrow bill[\ ]]$. It is also convenient to allow combinations of typing and complex O-terms. For instance, we could define

$$Q: \mathbf{p}[funA \rightarrow X: \mathbf{q}[setA \twoheadrightarrow \{Y\}], setA \twoheadrightarrow \{Z: \mathbf{r}\}]$$

as a syntactic sugar for

$$Q: \mathbf{p} \wedge Q[funA \rightarrow X[setA \twoheadrightarrow \{Y\}], setA \twoheadrightarrow \{Z\}] \wedge X: \mathbf{q} \wedge Z: \mathbf{r}.$$

Formulas that are "sugared" in this way will be used throughout this paper, especially in the examples.

2.2. *Semantics*

Given a language **L** of O-logic, a *semantic structure*, I, is a tuple $\langle U, I_{\mathbf{F}}, I_{\rightarrow}, I_{\twoheadrightarrow}, I_{\Sigma^*}, \leqslant_{\Sigma} \rangle$. Here U is a nonempty (possibly infinite) universe of all objects; the

[1] This flexibility, required by the object-oriented syntax, is the primary reason for choosing the name "term" for statements, such as O-terms, which will actually play the role of formulas in O-logic. Another reason is that, as we shall see shortly, id-terms can be viewed as a special case of complex O-terms. Our terminology in this respect is consistent with Maier's [43] and with other works in this field.

[2] These formulas are called "molecular" because, on one hand, they serve as building blocks for other O-formulas but, on the other hand, they are not really atomic. As we shall see later, a molecule may be equivalent to a conjunction of simpler O-terms.

mapping $I_{\mathbf{F}}$ interprets every k-ary object constructor $f \in \mathbf{F}$ as a total function $I_{\mathbf{F}}(f)$: $U^k \to U$. When $k = 0$, $I_{\mathbf{F}}$ can be viewed as a mapping from the constants of $\mathbf{L}$ into U.

The mapping $I_{\rightarrow}$ interprets each single-valued attribute $funA \in \mathbf{A}_{\rightarrow}$ as a *partial* function $I_{\rightarrow}(funA)$: $U \to U$. Similarly, $I_{\twoheadrightarrow}$ maps every set-valued attribute $setA \in \mathbf{A}_{\twoheadrightarrow}$ into a *partial* function $I_{\twoheadrightarrow}(setA)$: $U \to 2^U$, where 2^U denotes the power-set of U. The mapping I_{Σ} interprets each class-name in Σ as a subset of U, that is, I_{Σ}: $\Sigma \to 2^U$. Finally, $\leqslant_{\Sigma}$ is a transitive and reflexive relation (a pre-order) on Σ such that if $\mathbf{p} \leqslant_{\Sigma} \mathbf{q}$ then $I_{\Sigma}(\mathbf{p}) \subseteq I_{\Sigma}(\mathbf{q})$ (i.e., $\leqslant_{\Sigma}$ models the subclass relationship).

As in the classic predicate calculus, a semantic structure should be viewed as a possible world that gives an interpretation to the alphabet symbols and formulas of $\mathbf{L}$. For instance, the elements of U play the role of "real" objects in the possible world I. In contrast, the elements of $\mathbf{F}^*$ (the set of all variable-free id-terms) are the *denotations* or "names" given (via $I_{\mathbf{F}}$) to some of these objects. A semantic structure, I, interprets the typing O-terms and complex O-terms as assertions about properties of the named objects in U. Intuitively, an O-term t is satisfied by a semantic structure I, if the properties asserted in t hold for the object of U referred to by the oid part of t. Thus, if no attribute is explicitly listed in a complex O-term (e.g., $a[\ ]$) then this O-term is satisfied in every structure. The formal development, below, generally follows the terminology of [22], adapted to O-logic.

A variable-assignment, v, is a mapping $\mathbf{V} \to U$. We extend it to id-terms in the usual way: $v(f(..., T, ...)) = I_{\mathbf{F}}(f)(..., v(T), ...)$ (if f is 0-ary then we have $v(f) = I_{\mathbf{F}}(f)$). For convenience, we also extend v to complex O-terms so that $v(T[...])$ is defined as $v(T)$.

Given a semantic structure I and a variable assignment v, we can talk about formulas satisfied by I with respect to v. Formally, formula satisfaction, denoted $I \models_v \phi$, is defined as

- For a typing O-term $\phi \equiv T : \mathbf{p}$, $I \models_v \phi$ if and only if $v(T) \in I_{\Sigma}(\mathbf{p})$.
- For an is-a O-term $\phi \equiv \mathbf{p} : \mathbf{q}$, $I \models_v \phi$ if and only if $\mathbf{p} <_{\Sigma} \mathbf{q}$.
- For a complex O-term $\phi \equiv T[..., funA_i \to T_i, ..., setA_j \twoheadrightarrow \{S_1, ..., S_m\}, ...]$, $I \models_v \phi$ if and only if the following conditions hold:

— For each functional attribute $funA_i$, $I_{\rightarrow}(funA_i)(v(T)) = v(T_i)$ and $I \models_v T_i$.

— For each set-valued attribute $setA_j$, $\{v(S_1), ..., v(S_m)\} \subseteq I_{\twoheadrightarrow}(setA_j)(v(T))$ and $I \models_v S_k$, for each $k = 1, ..., m$.

Observe that the semantics of set-valued attributes is defined in such a way that if, say, $I \models_v t[setA \twoheadrightarrow \{a_1, ..., a_k\}]$ then $I \models_v t[setA \twoheadrightarrow \{a_{i_1}, ..., a_{i_l}\}]$, for every subset $\{a_{i_1}, ..., a_{i_l}\} \subseteq \{a_1, ..., a_k\}$. In particular, $I \models_v t[setA \twoheadrightarrow \{\ \}]$ holds. However, in general, I does not need to satisfy $t[setA \twoheadrightarrow \{\ \}]$, since $I_{\twoheadrightarrow}(setA)$ is a partial function and thus it may be undefined on $v(t)$.

We also remark that $I \models_v \mathbf{p} : \mathbf{q}$ implies $I_{\Sigma}(\mathbf{p}) \subseteq I_{\Sigma}(\mathbf{q})$. However, the latter does not

imply the former. The definitions were set up this way because if $I_\Sigma(\mathbf{p}) \subseteq I_\Sigma(\mathbf{q})$ were entailing $I \models_v \mathbf{p} : \mathbf{q}$ then O-logic would not have had a complete proof theory. Indeed, it is easy to show that if $I_\Sigma(\mathbf{p}) \subseteq I_\Sigma(\mathbf{q})$ were implying $I \models_v \mathbf{p} : \mathbf{q}$ then the so called *query containment problem* (known to be co-recursively enumerable [48]) could then be reduced to the question of whether some set of O-formulas logically implies $\mathbf{p} : \mathbf{q}$. We also note that, according to our semantics, an object may belong to no class, as $\bigcup_{\mathbf{p} \in \Sigma} I_\Sigma(\mathbf{p})$ does not have to be equal to U.

For O-formulae ϕ and ψ, the meaning of $\phi \wedge \psi$, $\phi \vee \psi$, and $\neg\phi$ is defined in the usual way. The meaning of a quantified O-formula is also standard: $I \models_v \psi$, where $\psi = (\forall X)\phi$ (resp., $\psi = (\exists X)\phi$) if for every (resp., some) μ that agrees with v everywhere except possibly X, $I \models_\mu \phi$. Clearly, if ϕ is a *closed* O-formula (no free variables), its meaning is independent of a variable assignment, and we can write $I \models \phi$, omitting v. A semantic structure I is a *model* of ϕ if $I \models \phi$, in which case we say that I *satisfies* ϕ.

2.3. *Predicates as "Value-Based" Objects*

It has been argued (e.g., [4, 16, 31]) that in certain cases it may be advantageous to have predicates on a par with objects. A typical situation where this might be needed arises when one has to assert certain symmetric relationships among objects (e.g., equality, adjacency, etc.). Although, as we shall see, any such relationship can always be encoded as an object, this may not be the most natural way to go. In this section, we first show how predicates can be encoded as O-terms and then present a direct semantics that does not appeal to this encoding.

First-order predicates can be mapped into O-logic as follows: If $p(T_1, ..., T_n)$ is an atomic formula in predicate calculus, then the corresponding O-term is $f_p(T_1, ..., T_n) : \mathbf{p}[arg_1 \rightarrow T_1, ..., arg_n \rightarrow T_n]$, where f_p is a new function symbol and $\mathbf{p}$ is a new class name; both f_p and $\mathbf{p}$ uniquely correspond to p and are specifically chosen for this particular encoding. Note that every ground fact of the form $p(t_1, ..., t_n)$ corresponds to a unique object with the id $f_\mathbf{p}(t_1, ..., t_n)$, which makes the above O-term a "valued-based" construct in the sense of [52] (i.e., this object's identity is totally dependent on the values of the attributes).

The mapping between atomic formulas of the first-order predicate calculus and O-terms described above allows us to use object identities, complex ojects, and predicates in the same framework. However, we prefer to forego the mental exercise of encoding predicates as objects, and show how predicates can be incorporated into O-logic directly.

First, we extend the alphabet with a set of *predicate symbols*, Π, that is disjoint from $\mathbf{F}$, $\mathbf{A}_{\rightarrow}$, $\mathbf{A}_{\twoheadrightarrow}$, Σ, and $\mathbf{V}$. The language is then enriched with a new kind of molecular formulae, *predicate terms* (abbr. *P-terms*), which are statements of the form $p(T_1, ..., T_k)$, where p is a k-ary predicate symbol and the T_i's are complex O-terms. The notion of O-formula is now extended appropriately to include P-terms. Note that by extrapolating the syntactic tradition of object-oriented languages, we introduced P-terms in a slightly more general form than it is customary for atomic

formulas in predicate calculus. For instance, *likes*(*mary*, *john*[*salary* → 99*K*]) is a syntactically correct predicate term.

As mentioned in Section 2.1, to guard against the possibility of syntactic confusion between predicates and id-terms, we do not allow dropping the brackets "[ ]" in complex O-terms of the form *t*[ ], when these appear as molecular formulas. However, writing *mary*[*likes* → *father*(*john*)] instead of *mary*[*likes* → *father*(*john*[ ]] is permitted (and, in fact, encouraged), because here *father*(*john*) cannot be confused with a P-term, since the latter cannot occur inside an O-term.

The semantics of O-logic is extended to accommodate P-terms in an obvious way: Semantic structures now have the form $I = \langle U, I_{\mathbf{F}}, I_{\rightarrow}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi} \rangle$, where U, $I_{\mathbf{F}}$, $I_{\rightarrow}$, $I_{\twoheadrightarrow}$, I_{Σ},and $\leqslant_{\Sigma}$ are as before and I_{Π} interprets each k-ary predicate symbol $p \in \Pi$ by a relation $I_{\Pi}(p) \subseteq U^k$. Satisfaction of P-terms is defined as follows: Given a semantic structure I and a variable assignment v,

- $I \models_v p(T_1, ..., T_k)$ if and only if $\langle v(T_1), ..., v(T_k) \rangle \in I_{\Pi}(p)$ and $I \models_v T_i$.

As usual, the equality predicate "=" gets special treatment:

- $I_{\Pi}(=) \stackrel{\text{def}}{=} \{\langle u, u \rangle \mid u \in U\}$,

which is equivalent to saying that for any pair of id-terms, T and S,

- $I \models_v (T = S)$ if and only if $v(T) = v(S)$.

The reader can easily verify that this semantics coincides with the one that results from the encoding introduced earlier; it is also consistent with the usual semantics of atomic formulas in predicate calculus.

3. Databases and Queries

A *deductive database* (or a *logic program*—we will use these two terms interchangeably) is a set of closed O-formulae. If **P** is a set of O-formulae and ϕ is an O-formula, we write $\mathbf{P} \models \phi$ if and only if ϕ is true in every model of **P**. In this case, we also say that ϕ is *logically implied* (or *entailed*) by **P**.

Given an O-logic language **L**, let **V** be the set of variables in the alphabet of **L** and let **F** be the set of object constructors. A *substitution* is a mapping $\sigma: \mathbf{V} \rightarrow \{$id-terms of $\mathbf{L}\}$ that is an identity everywhere outside a finite set $\mathrm{dom}(\sigma) \subseteq \mathbf{V}$, called the *domain* of σ. Any substitution can be extended to id-terms by letting it commute with constructors, as usual. Substitutions are further extended to O-terms and quantifier-free O-formulae by letting them commute with logical connectives and defining $\sigma(T[..., funA_i \rightarrow T_i, ..., setA_j \twoheadrightarrow \{S_1, ..., S_k\}, ...]) = \sigma(T)[..., funA_i \rightarrow \sigma(T_i), ..., setA_j \twoheadrightarrow \{\sigma(S_1), ..., \sigma(S_k)\}, ...]$.

If ϕ is a quantifier-free O-formula and σ is a substitution, the formula $\sigma(\phi)$ is an *instance* of ϕ. A formula or a term is *ground* if it has no variables. A substitution σ is *ground* if for each $X \in \mathrm{dom}(\sigma)$, $\sigma(X)$ is a ground id-term. For any pair of substitutions θ and μ, $\theta \circ \mu$ denotes their *composition*, that is, $\theta \circ \mu(X) = \theta(\mu(X))$.

A *query* is a statement of the form $?-Q$, where Q is an O-term. Given a database **P**, the set of *answers* to $?-Q$ is the smallest set of ground O-terms, that is,

(1) closed under "$\models$" and

(2) contains all instances of Q logically entailed by **P**.

The first condition here is a mere technicality, motivated by the fact that in O-logic sets of O-terms may imply other O-terms in a non-trivial way. For instance, in the absence of this condition, the query

$$?-john[children \twoheadrightarrow \{X\}]$$

to the database consisting of

$$john[children \twoheadrightarrow \{sally, bob\}]$$

would have had the answer $john[children \twoheadrightarrow \{sally\}]$ and $john[children \twoheadrightarrow \{bob\}]$, but not $john[children \twoheadrightarrow \{sally, bob\}]$. Condition (1) helps eliminate this anomaly.

Unlike the original Maier's proposal, extensional databases are *not* identical to semantic structures in our formalism. For us, an *extensional database* is a finite set of O-terms and P-terms (not necessarily ground, if one so wishes). However, in Section 6 we shall see that every extensional database **P** can be *associated* with a certain "canonic" semantic structure that satisfies every molecule in **P**.

Horn rules are defined in the standard way, as implicitly universally quantified statements of the form $\tau \Leftarrow \tau_1 \,\&\, \cdots \,\&\, \tau_n$, where τ and the τ_i's are positive O-terms or P-terms. For instance, the rule $M[works \rightarrow D] \Leftarrow D : \mathbf{dept}[mngr \rightarrow M : \mathbf{empl}]$ states that department managers work in the departments they manage. Here **dept** and **empl** are classes and M, D are object variables. Universal quantifiers over M and D are omitted.

4. Examples

In this section, all logic programs consist of Horn clauses only and therefore the logical implication "$\models$" of Section 2.2 is quite adequate for understanding the examples below. An O-logic adaptation of the classic theory of logic programs is outlined in Section 10.

4.1. *A Warm-up*

We start by casting a simple relational database into the object-oriented syntax of O-logic. We remind that according to Section 2.3, every relational database is also a syntactically correct O-logic extensional database. The next example, however, shows a different representation for a familiar problem—one that is in line with the object-oriented paradigm. Whether one likes the relational syntax or the object-oriented one, both representations are within the realm of O-logic.

EXAMPLE 4.1. The common Employee–Department–Manager example has the following object-oriented look:

$$john : \mathbf{empl}[works \rightarrow cs : \mathbf{dept},\ name \rightarrow \text{"}john\text{"} : \mathbf{string}]$$

$$cs : \mathbf{dept}[mngr \rightarrow john : \mathbf{empl},\ name \rightarrow \text{"}cs\text{"} : \mathbf{string},\ budget \rightarrow 1000 : \mathbf{int}]$$

$$ee : \mathbf{dept}[name \rightarrow \text{"}ee\text{"} : \mathbf{string},\ budget \rightarrow 1000 : \mathbf{int}].$$

Here, *john*, *cs*, and *ee* are constants denoting real-world objects (presumably, someone called John and some concrete computer science and electrical engineering departments). Likewise, the constants "*john*," "*ee*," and "*cs*" denote abstract objects that are strings of characters; in our example, these strings happen to be the values of the *name*-attribute in the context of the objects *john*, *ee*, and *cs*, respectively. The above O-terms also classify *john* as an instance of the class **empl**, *cs* and *ee* as instances of **dept**, "*john*," "*cs*" as instances of the class **string**, and 1000 as an **int**. By the way, 1000 is a constant symbol denoting an abstract object, the number "Thousand."

In O-logic, objects that are indistinguishable via the values of their attributes can be distinguished by their identities. Explicit manipulation with object ids gives us added flexibility in writing queries. The next example shows how this enables explicit control over elimination of duplicates. Depending on whether or not duplicates' elimination is desirable, the query asking for a list of all budgets can be written either as

$$no\_duplicates(Y)[budget \rightarrow Y] \Leftarrow X : \mathbf{dept}[name \rightarrow Z : \mathbf{string},\ budget \rightarrow Y : \mathbf{int}],$$

or as

$$duplicates\_welcome(X)[budget \rightarrow Y] \Leftarrow X : \mathbf{dept}[name \rightarrow Z : \mathbf{string},\ budget \rightarrow Y : \mathbf{int}],$$

where *no_duplicates* and *duplicates_welcome* are some unary function symbols. In the first case, the identity of newly created objects depends on budget only. Thus, duplicate tuples containing the same numerical values of the budget are eliminated, since they correspond to the same object. In contrast, the second rule retains duplicates, since the id here depends on the department rather than the budget.

4.2. *The Problem of Existential Variables*

Let us now turn to the issue of anomalies exhibited by the logic originally proposed in [43].

EXAMPLE 4.2. Consider a slightly simplified version of the "interesting pair" example from [43]:

$$P[emp \rightarrow E,\ mngr \rightarrow M] \Leftarrow E : \mathbf{empl}[name \rightarrow N : \mathbf{string},\ works \rightarrow D : \mathbf{dept}[mngr \rightarrow M : \mathbf{empl}[name \rightarrow N]]]. \quad (1)$$

This rule is supposed to construct objects representing pairs employee–manager such that the employee's department's manager's name coincides with the employee's name.

As observed in [43], universal quantification over the object-variable P does not make sense in this example. It was then suggested that P should be quantified existentially. More precisely, Maier suggested the following quantification for the above rule: $(\forall E\ \forall M\ \forall N\ \exists P)$.

Unfortunately, the argument in [43] is not well substantiated, and the reasons for the choice of this particular quantification are not quite clear. In fact, we claim that the intended quantification here is $(\forall E\ \forall M\ \exists P\ \forall N)$ and Maier's quantification happens to work only because E and M functionally determine N. Should the attribute *name* be set-valued, the two quantifications would have different meanings.

There seems to be no obvious way of choosing between the above two quantifications solely on the basis of the syntactic structure of rule (1). The root of the problem is that the object variable P does not appear in the body of (1), leaving the quantification of P open to debates. Since [43] did not have object constructors, Maier's original logic had no way of connecting P to object variables in the body of (1), which led to the aforesaid *ad hoc* choice of quantification. A better representation for the problem at hand would be:

$$\begin{aligned} &\mathit{namesake}(E, M)[\mathit{emp} \rightarrow E, \mathit{mngr} \rightarrow M] \Leftarrow \\ &\quad E : \mathbf{empl}[\mathit{name} \rightarrow N : \mathbf{string}, \\ &\qquad \mathit{works} \rightarrow D : \mathbf{dept}[\mathit{mngr} \rightarrow M : \mathbf{empl}[\mathit{name} \rightarrow N]]], \end{aligned} \tag{2}$$

where *namesake* is an object constructor. Unlike (1), the semantics of (2) is well defined and corresponds to the intended meaning.

The constructor *namesake* can be viewed as a Skolem function arising from the quantification $(\forall E\ \forall M\ \exists P\ \forall N)$ mentioned earlier. The explicit use of object constructors in our version of O-logic is a way of disambiguating the quantification.

EXAMPLE 4.3. Consider a graph stored as a set of objects of the form $E : \mathbf{edge}[\mathit{start} \rightarrow X : \mathbf{node}, \mathit{end} \rightarrow Y : \mathbf{node}]$. The set of (not necessarily simple) paths between nodes in the graph can be specified via the following rules:

$$\begin{aligned} &E : \mathbf{path} \Leftarrow E : \mathbf{edge}[\mathit{start} \rightarrow X : \mathbf{node}, \mathit{end} \rightarrow Y : \mathbf{node}] \\ &\mathit{add}(E, P) : \mathbf{path}[\mathit{start} \rightarrow X, \mathit{end} \rightarrow Y] \Leftarrow \\ &\quad E : \mathbf{edge}[\mathit{start} \rightarrow X : \mathbf{node}, \mathit{end} \rightarrow Z : \mathbf{node}] \\ &\quad \&\ P : \mathbf{path}[\mathit{start} \rightarrow Z : \mathbf{node}, \mathit{end} \rightarrow Y : \mathbf{node}]. \end{aligned} \tag{3}$$

The first clause simply says that every edge is also a path. The second clause says that adding an edge adjacent to a path creates another, longer path. Thus, given

a query $?-P$: **path**[*start* → *X* : **node**, *end* → *Y* : **node**], the set of answers will be composed of O-terms of the form $add(e_1, add(e_2, ...e_k...))[start \rightarrow a, end \rightarrow b]$, where the e_j's are the object id's of edges and *a*, *b* are object id's of nodes. Here, for any path, its object id reconstructs the way this particular path has been built and provides a useful information in its own right. The answer-set to the above query will have an object for every path connecting every pair of nodes. Since non-simple paths are allowed, the answer to this query may be an infinite set of objects.

Should one be interested in the reachability relation between nodes rather than the all-paths problem, the corresponding set of rules will be

$$\begin{aligned} &reach(X, Y) : \mathbf{path}[start \rightarrow X, end \rightarrow Y] \Leftarrow \\ &\quad E : \mathbf{edge}[start \rightarrow X : \mathbf{node}, end \rightarrow Y : \mathbf{node}] \\ &reach(X, Y) : \mathbf{path}[start \rightarrow X, end \rightarrow Y] \Leftarrow \\ &\quad E : \mathbf{edge}[start \rightarrow X : \mathbf{node}, end \rightarrow Z : \mathbf{node}] \\ &\quad \& P : \mathbf{path}[start \rightarrow Z : \mathbf{node}, end \rightarrow Y : \mathbf{node}]. \end{aligned} \tag{4}$$

Here ids of the objects being constructed solely depend on the start and the end points of the respective paths. The set of answers to the aforementioned query will then be comprised of O-terms of the form $reach(a, b)[start \rightarrow a, end \rightarrow b]$. Consequently, the answer will contain exactly one object for each pair of nodes connected by a path.

It should be noted that (3) constructs *all* paths, including the non-simple ones and, therefore, an infinite number of them. If this is a bother, we could modify the second rule in (3) as follows:

$$\begin{aligned} &add(E, P) : \mathbf{path}[start \rightarrow X, end \rightarrow Y] \Leftarrow simple(add(E, P)) \\ &\quad \& E : \mathbf{edge}[start \rightarrow X : \mathbf{node}, end \rightarrow Z : \mathbf{node}] \\ &\quad \& P : \mathbf{path}[start \rightarrow Z : \mathbf{node}, end \rightarrow Y : \mathbf{node}], \end{aligned}$$

where *simple*(...) is a predicate that checks whether the id-term $add(E, P)$ represents a simple path. Because of the structure of the ids in (3), *simple*(...) can be easily defined in Prolog with negation (and hence in O-logic).

Example 4.3 not only demonstrates the flexibility of our logic, but also emphasizes once again the inadequacy of rule quantification suggested in [43]. Indeed, according to Maier's original proposal the second rule in (3) and (4) would have to be written as

$$\begin{aligned} &Path : \mathbf{path}[start \rightarrow X, end \rightarrow Y] \Leftarrow \\ &\quad E : \mathbf{edge}[start \rightarrow X : \mathbf{node}, end \rightarrow Z : \mathbf{node}] \\ &\quad \& P : \mathbf{path}[start \rightarrow Z : \mathbf{node}, end \rightarrow Y : \mathbf{node}], \end{aligned} \tag{5}$$

where *Path* is a new object variable. The last rule in (3) can be obtained from (5) by interpreting *add*(...) as a function that results from a Skolemization of *Path* with respect to the quantification ($\forall E\ \forall P\ \exists Path\ \forall X\ \forall Y\ \forall Z$), while the last rule in (4) is obtained from (5) by Skolemizing *Path* with respect to ($\forall X\ \forall Y\ \exists Path\ \forall E\ \forall P\ \forall Z$). However, there is nothing in the structure of the rule (5) that helps choose one quantification over the other. Beeri, Nasr, and Tsur [14, 15] espouse a view similar to that of [43], proposing another *ad hoc* choice of quantification. Our contention is that no single such decision regarding quantification should be hard-wired into the logic; instead, we contend that a variety of different quantifications should be expressible in a single formalism.

4.3. *Set-Grouping*

The next series of examples demonstrates our concept of grouping, compared to LDL. Example 4.4 uses grouping to express set-intersection and set-union operators (the latter is a built-in function in LDL). Unlike LDL, grouping in O-logic behaves monotonically and does not require stratification. For instance, the following O-logic program has a unique minimal model, provided that the domain and the interpretation of function symbols are fixed (e.g., as in Herbrand interpretations).

EXAMPLE 4.4. Set-intersection and union can be expressed as follows:

$$\begin{aligned} inter(S_1, S_2) : \mathbf{set}[elements \twoheadrightarrow \{\ \}] &\Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ &\quad \& S_2 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ inter(S_1, S_2)[elements \twoheadrightarrow \{X\}] &\Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{X\}] \\ &\quad \& S_2 : \mathbf{set}[elements \twoheadrightarrow \{X\}] \end{aligned} \tag{6}$$

$$\begin{aligned} union(S_1, S_2) : \mathbf{set}[elements \twoheadrightarrow \{\ \}] &\Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ &\quad \& S_2 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ union(S_1, S_2)[elements \twoheadrightarrow \{X\}] &\Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{X\}] \\ &\quad \& S_2 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ union(S_1, S_2)[elements \twoheadrightarrow \{X\}] &\Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{\ \}] \\ &\quad \& S_2 : \mathbf{set}[elements \twoheadrightarrow \{X\}]. \end{aligned} \tag{7}$$

Expressions of the form $elements \twoheadrightarrow \{X\}$ in the rule heads are examples of set-grouping in O-logic. The first clause in (6) says that if the *elements* attribute in both S_1 and S_2 is defined (and hence its value is a set that at least contains the empty set) then *elements* is also defined for the object $inter(S_1, S_2)$ that represents the intersection of sets represented via objects S_1 and S_2.

The second rule in (6) tells us that if X is an element of the set $S_1.elements$ (which here denotes the value of the attribute *elements* in the context of the object S_1) and, at the same time, it is a member of $S_2.elements$ then X also belongs to the set $inter(S_1, S_2).elements$. In other words, if *INTER* consists of the intersection-rules (6) plus some extensional database (e.g., $a : \mathbf{set}[elements \twoheadrightarrow \{c, d\}]$ and

$b : \mathbf{set}[elements \twoheadrightarrow \{d, e\}]$) then for any pair of **set**-objects f and g and any $h_1, ..., h_k \in \mathbf{F}^*$,

$$\begin{aligned} &INTER \models inter(f, g)[elements \twoheadrightarrow \{h_1, ..., h_k\}] \quad \text{if and only if} \\ &INTER \models f[elements \twoheadrightarrow \{h_1, ..., h_k\}] \wedge g[elements \twoheadrightarrow \{h_1, ..., h_k\}], \end{aligned} \tag{8}$$

which means that $inter(f, g)$ represents the intersection of sets represented by the objects f and g. Another way to say this is that in every model I of *INTER*, the set $I_{\twoheadrightarrow}(elements)(I_{\mathbf{F}}(inter)(I_{\mathbf{F}}(f), I_{\mathbf{F}}(g)))$ contains the intersection of the sets $I_{\twoheadrightarrow}(elements)(I_{\mathbf{F}}(f))$ and $I_{\twoheadrightarrow}(elements)(I_{\mathbf{F}}(g))$ and for some interpretations this containment turns into equality.

To see why (8) is a valid claim, consider an arbitrary model $I = \langle U, I_{\mathbf{F}}, I_{\rightarrow}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi} \rangle$ of *INTEGER*. The first rule in (6) guarantees that if $I_{\twoheadrightarrow}(elements)$ is defined on a pair of elements $u, v \in U$ then it is also defined on $I_{\mathbf{F}}(inter)(u, v)$. It is now seen from the definitions that for the second rule to be true, the following must hold in I:

$$I_{\twoheadrightarrow}(elements)(I_{\mathbf{F}}(inter)(u, v)) \supseteq I_{\twoheadrightarrow}(elements)(u) \cap I_{\twoheadrightarrow}(elements)(v).$$

This shows the "if" part of (8). For the "only if" direction, suppose that, say, $INTER \not\models f[elements \twoheadrightarrow \{q\}]$, for some $q \in \mathbf{F}^*$. It is then easy to construct an interpretation J such that $J \not\models inter(f, g)[elements \twoheadrightarrow \{q\}]$.

It is worth noting the essential role of the first rule in (6): If only the second rule in (6) were used to define the intersection, then the definition would have been correct only for intersecting non-empty sets, since X must be bound to something in order for the rule to fire. This also explains why it takes three rules in (7) to define set-union. Although having to deal with empty sets separately is a nuisance, the problem can be easily circumvented via a simple syntactic sugar. Intuitively, what we need here is a new kind of variable that can be found to "nothing" so that the same O-term could represent empty as well as non-empty sets. To this end, we can introduce new kinds of variables that are, say, prefixed with a "?"-symbol and then restrict their occurrence to set-constructs only (i.e., to the inside of the braces "{ }"). Every clause C containing a "?"-variable, e.g., $?X$, is semantically equivalent to a pair of clauses, C_1 and C_2, where C_1 is the same as C except that $?X$ is replaced by a *new* ordinary variable (e.g., X_{213}), and C_2 is obtained from C by deleting $?X$ altogether. This transformation must be subsequently performed for every "?"-variable. Semantically, these variables can be accommodated by changing the definition of variable assignments, $v : \mathbf{V} \rightarrow U$, letting them be undefined on some or all of the "?"-marked variables. Filling in simple details of this semantics is left to the reader. With this new notation, set-union can be defined in just one rule:

$$union(S_1, S_2) : \mathbf{set}[elements \twoheadrightarrow \{?X, ?Y\}] \Leftarrow S_1 : \mathbf{set}[elements \twoheadrightarrow \{?X\}] \;\&\; S_2 : \mathbf{set}[elements \twoheadrightarrow \{?Y\}].$$

The next pair of examples compares our concept of grouping to that of LDL [13] and COL [2]—the two recent languages that model sets using a second-order semantics. The following program from [13] does not have a unique minimal model in LDL, while the O-logic prgram with a similar reading does. Here, a model I of a set of formulas S is called *minimal* if and only if for every other model J of S, whenever $I \models \phi$ for a molecule ϕ then also $J \models \phi$.

EXAMPLE 4.5. Consider the following database written in the LDL notation:

$$p(\langle X \rangle) :- q(X)$$
$$q(2),$$

where "$\langle\ \rangle$" is the grouping operator of LDL. Intuitively, this rule should be read as follows: Collect all elements of q, group them into a set, make this set into an element of the universe, and let p be true of this element. One would thus expect the answer to the query $?-p(Y)$ be $p(\{2\})$. However, because of the second-order semantics, the notion of model-minimization in LDL is such that this program may have several minimal models and therefore $p(\{2\})$ is not considered as an answer according to the minimal-model semantics.

To circumvent this problem, LDL introduces the concept of "model dominance" on top of the notion of minimality, which significantly complicates the matters. In O-logic, the mere notion of minimality suffices to handle the problem correctly. The corresponding O-logic program is:

$$a : \mathbf{p}[group \twoheadrightarrow \{X\}] \Leftarrow Y : \mathbf{q}[arg \rightarrow X]$$
$$b : \mathbf{q}[arg \rightarrow 2]$$

and the answer to the query $?- Y : \mathbf{p}[group \twoheadrightarrow \{Z\}]$ is $a[group \twoheadrightarrow \{2\}]$, as intended.

Our next example shows that COL is somewhat restrictive since it cannot handle certain simple situations that are perfectly legal in O-logic.

EXAMPLE 4.6. The following database in the COL syntax [2] depicts persons and their hobbies:

$$person(peter,\ \{bridge\})$$
$$person(tom,\ \{chess,\ tennis\}).$$

Suppose that, in addition, it is known that every hobby of Peter is also a hobby of Tom:

$$person(tom,\ hobby(peter))$$
$$person(tom,\ hobby(tom)),$$

where *hobby* is a data-function defined in COL as follows:

$$Y \in hobby(X) :- person(X, Z)\ \&\ Y \in Z.$$

Intuitively, the intention here is that *person*(*tom*, {*bridge*, *chess*, *tennis*}) should be true. Rather surprisingly, this COL program turns out to be nonstratified (not even locally stratified) and therefore it has no meaning. Again, the problem can be traced to the second-order semantics of COL. In O-logic the corresponding representation is simpler and has a unique minimal Herbrand model (Herbrand models are introduced in Section 7):

$$peter : \textbf{person}[hobby \twoheadrightarrow \{bridge\}]$$
$$tom : \textbf{person}[hobby \twoheadrightarrow \{chess, tennis\}]$$
$$tom[hobby \twoheadrightarrow \{X\}] \Leftarrow peter[hobby \twoheadrightarrow \{X\}].$$

The first pair of clauses here describes the objects *peter* and *tom*. The last rule says that every hobby of *peter* is also a hobby of *tom*. Let *HOBBY* denote the above O-logic program. Following the line of reasoning of Example 4.4, the reader can verify that for every ground term t, $HOBBY \models peter[hobby \twoheadrightarrow \{t\}]$ implies $HOBBY \models tom[hobby \twoheadrightarrow \{t\}]$, as intended.

Observe that in Example 4.6, X does not have to be a "?"-variable, since there is no need to treat empty sets separately: if *peter* has no hobbies, the *hobby*-attribute may well be undefined for *tom*, which is perfectly acceptable for the problem at hand. However, it will not be a mistake if X is replaced by a "?"-variable, say *?Y*. The difference is that in the latter case, whenever *hobby* is defined for *peter* (even if *peter* has an empty set of hobbies), the *hobby*-attribute must be defined for *tom* (again, even though *tom* may have no hobbies to speak of). Although the distinction between undefinedness of a set-valued attribute and one having an empty set as a value may seem insignificant, this allows us to talk about "meaningless" attributes, that is, attributes that are inapplicable to certain objects. This idea is further explored in [34].

4.4. *Inheritance*

The following example demonstrates certain (admittedly limited) capabilities for modeling inheritance. A more elaborate treatment appears in [34].

EXAMPLE 4.7. Consider the following database:

$$john : \textbf{workstudy}$$
$$\textbf{workstudy} : \textbf{student}$$
$$\textbf{workstudy} : \textbf{employee}$$
$$X[duty \twoheadrightarrow \{homework\}] \Leftarrow X : \textbf{student}$$
$$Y[duty \twoheadrightarrow \{officework\}] \Leftarrow Y : \textbf{employee}.$$

Here, the rules define duties assigned to different types of people. All students have a *homework* duty and all employees have an *officework* duty. Since **workstudy** is a subclass of **student** and **employee**, *john* inherits properties of each of these classes. In particular, $john[duty \twoheadrightarrow \{homework, officework\}]$ logically follows from the

database. Indeed, it is easy to verify that *john* : **student** and *john* : **employee** both follow from the first three clauses. Therefore, to satisfy the two rules of the database, every semantic structure must satisfy $john[duty \twoheadrightarrow \{homework, officework\}]$.

5. Equality and Inconsistency

In knowledge bases, it is important to be able to say that a pair of different object-denotations, e.g., *father*(*john*) *and peter*, refer to essentially the same thing. In O-logic, this can be represented via an equation $father(john) = peter$.

Example 5.1. Suppose the database D is:

$john$: **person**

$peter$: **person**

$father(john) = peter$

$father(X)[children \twoheadrightarrow \{X\}] \Leftarrow X$: **person**.

The last clause says that every person X belongs to the set of *children* of $father(X)$. It is easy to see that $father(john)[children \twoheadrightarrow \{john\}]$ logically follows from D. The equality $father(john) = peter$ means that *peter* and $father(john)$ denote the same object. Thus, given a query $?-peter[children \twoheadrightarrow \{Z\}]$ ("who are Peter's children?"), the answer is $Z = john$.

As in predicate calculus, along with the added expressibility, equations introduce a number of computational difficulties. Classic logic programming sidesteps these difficulties by prohibiting equations to appear in the rule heads. This essentially amounts to building a theory around the so-called *freeness axioms* for equality, which state that two terms are equal if and only if the terms are syntactically identical. However, in O-logic, equality is omnipresent; it sneaks in even when it is not mentioned explicitly. Indeed, functional attributes are interpreted as partial functions from objects to objects and so whenever a functional attribute is multiply defined on the same object, an equation ensues. For instance, if $obj[attr \rightarrow f]$ and $obj[attr \rightarrow g]$ are both in $\mathbf{P}$, then $f = g$ is logically entailed (check!). However, if also $\mathbf{P} \models (f \neq g)$ holds, then $\mathbf{P}$ has no model and inconsistency arises.

The potential for inconsistency pointed out above is rather unpleasant, for it means that one may be doing extensive computation just to find out that the database has a flaw and therefore all the answers produced so far make no sense. There are several different ways to handle this problem. In the earlier version of O-logic [31], a special constant $\top$ was reserved to represent inconsistency, and whenever something like $obj[attr \rightarrow f]$ and $obj[attr \rightarrow g]$ were derived, the O-term $obj[attr \rightarrow \top]$ was logically entailed. The important property of this semantics was that such inconsistency only affected this particular attribute, *attr*, and had no effect on other "good" attributes. This allowed us to localize the effect of inconsistency

and provided sound model-theoretic foundations for the resulting computation. Unfortunately, the way inconsistency-tolerance was handled in [31] was not quite satisfactory, since it relied on the freeness axioms for equality and thus limited the domain of applicability of O-logic by ruling out statements such as *father*(*john*) = *peter*.

In the end, we opted for a different semantics for functional attributes, the one given in Section 2.2. Nevertheless, inconsistency-tolerant functional attributes can be introduced in the following simple way: Let $\mathbf{A}_{\rightsquigarrow}$ be a new type of functional attributes, *inconsistency-tolerant*, whose semantics is similar to that of set-valued attributes, except that they obey the following pair of axiom chemata: For every $attr \in \mathbf{A}_{\rightsquigarrow}$,

$$\begin{aligned} T[attr \rightsquigarrow \top] &\Leftarrow T[attr \rightsquigarrow X] \,\&\, T[attr \rightsquigarrow Y] \,\&\, X \neq Y \\ T[attr \rightsquigarrow X] &\Leftarrow T[attr \rightsquigarrow \top]. \end{aligned} \tag{9}$$

Here, $attr \rightsquigarrow \ldots$ is a new piece of syntax that is used to distinguish inconsistency-tolerant functional attributes from the rest. The symbol "$\top$" is a distinguished constant in the alphabet of the language that represents inconsistent attribute values. The first axiom above says that if *attr* has two different values in the context of the same object then the value of this attribute should be declared as inconsistent (for this particular object). The second axiom says that if the value of an attribute is inconsistent then the attribute has *all* values at once. Semantically, attributes in $\mathbf{A}_{\rightsquigarrow}$ can be accommodated by augmenting every semantic structure, $I = \langle U, I_{\mathbf{F}}, I_{\rightarrow}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi} \rangle$, with an additional mapping, $I_{\rightsquigarrow}$, that interprets every inconsistency-tolerant attribute $attr \in \mathbf{A}_{\rightsquigarrow}$ as a partial function $I_{\rightsquigarrow}(attr)\colon U \to 2^U$. This mapping has the following properties: For every $u \in U$ for which $I_{\rightsquigarrow}(attr)(u)$ is defined,

(1) $I_{\rightsquigarrow}(attr)(u)$ is either a singleton set $\{v\}$, for some $v \in U$, or $I_{\rightsquigarrow}(attr)(u) = U$; and

(2) if $I_{\mathbf{F}}(\top) \in I_{\rightsquigarrow}(attr)(u)$ then $I_{\rightsquigarrow}(attr)(u) = U$.

With this semantics, if *john*[*salary* $\rightsquigarrow$ 99*K*, *age* $\rightsquigarrow$ 40] and *john*[*age* $\rightsquigarrow$ 50] both hold true and $40 \neq 50$ is known, then *john*[*salary* $\rightsquigarrow$ 99*K*, *age* $\rightsquigarrow \top$] and *john*[*age* $\rightsquigarrow$ 120] logically follow, while *john*[*salary* $\rightsquigarrow$ 80*K*] does not, unless *salary* becomes inconsistent (for a different reason). Whether or not the information about John's salary is inconsistent, can be tested by asking the query *?*—*john*[*salary* $\rightsquigarrow \top$]. Of course, since the logical entailment "$\models$" is only semi-decidable (Section 9.6), evaluation of this query may not terminate in the lucky case when John's salary is consistent.

The difference between the inconsistency-tolerant attributes in [31] and the present version is that now the freeness axioms are not built into the semantics. Rather, we can have an arbitrary equation theory, *EQ*, and derive inequality using, say, Reiter's closed world assumption (CWA), that is, by postulating that $EQ \mid\!\approx (t \neq s)$ if $EQ \not\models (t = s)$. Here "$\mid\!\approx$" denotes logical implication relatively to

CWA; it will be further discussed in Section 10.3. In our example, closed world assumption sanctions the derivation of the inequality $40 \neq 50$, provided that the programmer-supplied equational theory does not insist that $40 = 50$.[3]

The above semantics for the attributes in $\mathbf{A}_{\rightsquigarrow}$ is *monotonic* in the sense that the addition of new facts and rules to the database does not invalidate the information that was there originally. In particular, this semantics does not preclude conclusions drawn from inconsistent data. For instance, in the above example, if $john[age \rightsquigarrow \top]$ were derived and the database had a rule $john[category \rightsquigarrow \text{"}senior\text{"}] \Leftarrow john[age \rightsquigarrow 80]$ then the rule would still fire, yielding the conclusion that *john* is a senior citizen. In [33, 35], the notion of *epistemic* entailment was introduced, which guards against derivations of this sort. Epistemic entailment can be adapted to O-logic, but this issue is beyond the main focus of the present paper and will not be discussed here.

6. Discussion

At this point it is instructive to make several observations regarding the differences between our version of O-logic and Maier's original work. One of the main distinctions is that object constructors are explicitly part of our language. This seemingly minor difference is responsible for the elimination of most of the problems that plagued [43].

Among the less significant differences is the fact that we do not distinguish between so-called atomic data values and other objects. For us, each atomic value (e.g., a numeral) is both an object and its own object id. Furthermore, unlike [43], we do not find it useful to restrict the semantics so that atomic objects will have no internal states (in [43], attributes are undefined on atomic objects). In contrast, in our setting, atomic values may have properties (e.g., $3[divisibility \rightarrow prime, type \rightarrow int]$, $\text{"}john\text{"}[type \rightarrow string, length \rightarrow 4]$), just like LISP atoms do. Our choice is based on the observation that Maier's restriction on atomic objects is not essential, and we are not at all sure that atomicity is a useful notion in this context.

Our treatment of sets is another important extension of [43]. The interesting comparison here is with the treatment of sets in LDL [13], COL [2], and ELPS [39]. LDL is an ambitious attempt to introduce sets into logic programs in a clean way. Unfortunately, LDL turned out to be far too expressive than what one may hope to be able to compute. As shown in [13], the following program is tantamount to the famous set-theoretic paradox:

$$p(X)$$
$$p(\langle X \rangle) \leftarrow p(X).$$

According to the semantics of LDL, p would have to contain the set of all sets as an element, which causes the paradox. The possibility of paradoxes in LDL stems

[3] Of course, in a real programming language, integers will come with a built-in theory of equality, so that it will not be possible to say that $40 = 50$.

from its untyped second-order semantics in which variables may range over sets as well as individuals; "untyped" means that in the course of a computation the *same* variable may become bound to sets as well as individuals. In contrast, the semantics of O-logic is first order[4] and variables cannot range over sets. Sets in the semantic domain can be indirectly represented via object ids, but are not explicitly present here. In [18] it is shown that first-order semantics not only precludes paradoxes, but also makes sets computationally more tractable.

Sets in O-logic are flat in the sense that a set may contain only object ids as members, but not other sets. However, since any id may represent some other set, we are able to model sets of arbitrary nesting depth. We contend that this representation is essentially all one needs to have for computing with sets. Moreover, representing sets via ids has certain computational advantages, since in most cases one only needs to verify that, say, two sets are *intentionally* equal, that is, have equal ids. In this situation, LDL and other related identity-less formalisms will require a proof that the *definitions* of the two sets always yield the same *extension*, which is not even recursively enumerable.

At the other end of the spectrum, COL and ELPS take a conscious decision to exclude infinite sets from consideration. Although this restriction helps to avoid many of the semantic problems of LDL, it is somewhat artificial and limits the expressiveness of these languages. For instance, even though a set might be infinite, we still may want to *reason* about it by manipulation with its object id (which is what mathematicians often do). It also makes perfect sense to check if certain element is a member of a potentially infinite set. In COL and ELPS, such programs do not even have a meaning. Furthermore, for any sufficiently expressive language (e.g., Horn clauses with function symbols) the question of whether a set defined by a set of rules is finite, is undecidable (e.g., [48]) and thus—even for Horn clauses—the user cannot be sure whether his program makes any sense to a system that outlaws infinite sets. We believe that in O-logic a better balance is struck between expressiveness and computability: unlike COL and ELPS, O-logic allows (albeit indirectly) arbitrarily deeply nested and infinite sets, but the semantic problems of LDL do not arise. In addition, the second-order semantics of LDL, COL, and ELPS heavily taxes these languages by requiring stratification with respect to their grouping mechanisms, even when programs are Horn.

Our notion of a complex object is more flexible, too (cf. [1, 28]). It is usually assumed that complex objects are constructed by means of tuple and set constructors. However, O-terms combine the two constructs in one, creating a new kind of object. For instance, $children(john)[father \rightarrow john,\ kids \twoheadrightarrow \{bill,\ mary\}]$ is structurally different from $john[name \rightarrow \text{"}John\text{,"}\ \ kids \rightarrow children(john)[kids \twoheadrightarrow \{bill,\ mary\}]]$. The latter can be obtained by first applying the set constructor and then the tuple constructor, while no such sequence of operations can yield the former O-term. Indeed, the second component of the former O-term (namely $\{bill,\ mary\}$) is not an O-term by itself (it has no associated id), and therefore cannot be

[4] See [18, 20] for more discussion on the higher-order vs first-order semantics.

constructed independently from the object *children*(*john*). This corresponds to *weak entities* in the entity–relationship model.

Finally, we note that O-terms can denote cyclic objects, although this feature is not new and can be found in other formalisms (e.g., [5, 14, 38]). However, it should be noted that in [38] cyclic objects cannot be constructed by a query, since LDM bans cyclic queries. We do not know if such objects can be constructed in [14], but they certainly can be in O-logic, e.g., $X[self \rightarrow X] \Leftarrow X$.

7. Relationship to Predicate Calculus

We have seen that ordinary predicate calculus is a proper subset of O-logic. The following result shows that, nevertheless, no additional expressive power has been gained:

THEOREM 7.1. *Let* **L** *be an O-logic language. Then there is a language* **L**′ *of the first-order predicate calculus and a pair of transformations*

Φ: *O-logic formulas in* **L** $\rightarrow$ *Predicate-calculus well-formed formulas in* **L**′

Ψ: *O-logic semantic structures for* **L** $\rightarrow$ *Predicate-calculus semantic structures for* **L**′

such that for every O-formula ϕ *and semantic structure I for* **L**, $I \models_{\mathrm{O}} \phi$ *if and only if* $\Psi(I) \models_{\mathrm{PC}} \Phi(\phi)$. *Here* "$\models_{\mathrm{O}}$" *denotes the logical entailment relation of Section* 2.2 *and* "$\models_{\mathrm{PC}}$" *is the logical entailment relation of classical predicate calculus.*

Proof. Let the alphabet of **L** consist of **F**, $\mathbf{A}_{\rightarrow}$, $\mathbf{A}_{\twoheadrightarrow}$, Σ, Π, **V**, connectives, quantifiers, etc. The corresponding language **L**′ for predicate calculus has the same set of variables **V**, connectives, quantifiers, and other paraphernalia. The set **F**′ of function symbols in **L**′ consists of **F** and Σ. The elements of Σ are viewed as constants in **L**′, while the elements of **F** retain the arity they had in **L**. The set Π' of predicate symbols in **L**′ contains the sets Π, $\mathbf{A}_{\twoheadrightarrow}$, $\mathbf{A}_{\rightarrow}$, and a triple of new predicates: "*object*," "*instance*," and "*subclass*." Arities of the members of Π' that come from Π are the same as they were in **L**; *object* is a unary predicate and the rest of the predicates are binary.

The mapping Φ transforms every typing O-term $T : \mathbf{c}$ into *instance* $(T, \mathbf{c})$ and every is-a O-term $\mathbf{p} : \mathbf{q}$ into *subclass* $(\mathbf{p}, \mathbf{q})$. Φ leaves P-terms essentially intact, except in the case when a P-term contains complex O-terms inside. More precisely, $p(T_1, ..., T_k)$ is transformed into $p(\bar{T}_1, ..., \bar{T}_k) \wedge \Phi(T_1) \wedge \cdots \wedge \Phi(T_k)$, where $\bar{T}_i$ is the id-term representing the object id of T_i (i.e., $T_i = \bar{T}_i[...]$), $i = 1, ..., k$.

A complex O-term $T[..., funA \rightarrow R, ..., setA \twoheadrightarrow \{S_1, ..., S_l\}, ...]$ is transformed by Φ into $\cdots \wedge funA(T, \bar{R}) \wedge \Phi(R) \wedge \cdots \wedge setA(T, \bar{S}_1) \wedge \Phi(S_1) \wedge \cdots \wedge setA(T, \bar{S}_l) \wedge \Phi(S_l) \wedge \cdots$, where $\bar{R}$ and $\bar{S}_i$, $i = 1, ..., l$, are the terms occurring in the object id position of the complex O-terms R and S_i, respectively. Complex O-terms of the

form $T[\ ]$ are transformed into a tautology, e.g., **true**. We also need one Horn rule per functional attribute to express the functionality property and two rules to express transitivity of the is-a and typing O-terms. Finally, quantifiers are relativized as follows: $\exists X\phi$ is converted into $\exists X(object(X) \wedge \phi)$ and $\forall X\phi$ into $\forall X(object(X) \rightarrow \phi)$.

The mapping Ψ is defined in an obvious way: Let $I = \langle U, I_{\mathbf{F}}, I_{\rightarrow}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi}\rangle$ be a semantic structure for **L**. The corresponding semantic structure $\Psi(I) = I' \stackrel{\text{def}}{=} \langle U', I'_{\mathbf{F}'}, I'_{\Pi'}\rangle$ is such that:

- $U' \stackrel{\text{def}}{=} U \cup \Sigma$ (a disjoint union);
- The interpretation of function symbols $I'_{\mathbf{F}'}$ is defined as follows:

For each $f \in \mathbf{F}$, $I'_{\mathbf{F}'}(f) = I_{\mathbf{F}}(f)$ over the arguments in U; $I'_{\mathbf{F}'}(f)$ is defined arbitrarily if at least one of its arguments is in $U' - U$. For every $\mathbf{p} \in \Sigma$, $I'_{\mathbf{F}'}(\mathbf{p})$ is defined to be $\mathbf{p}$.

- $I'_{\Pi'}$, the interpretation of the symbols in Π', coincides with I_{Π} on the elements of Π; on the rest of the symbols in Π', the mapping $I'_{\Pi'}$ is defined as follows:

  — $I'_{\Pi'}(object) \stackrel{\text{def}}{=} U$;
  — $I'_{\Pi'}(instance) \stackrel{\text{def}}{=} \{\langle u, \mathbf{p}\rangle \mid u \in I_{\Sigma}(\mathbf{p})\}$;
  — $I'_{\Pi'}(subclass) \stackrel{\text{def}}{=} \{\langle \mathbf{p}, \mathbf{q}\rangle \mid \mathbf{p} \leqslant_{\Sigma} \mathbf{q}\}$;
  — $I'_{\Pi'}(funA) \stackrel{\text{def}}{=} \{\langle u, v\rangle \mid v = I_{\rightarrow}(funA)(u)\}$, for each $funA \in \mathbf{A}_{\rightarrow}$;
  — $I'_{\Pi'}(setA) \stackrel{\text{def}}{=} \{\langle u, v\rangle \mid v \in I_{\twoheadrightarrow}(setA)(u)\}$, for each $setA \in \mathbf{A}_{\twoheadrightarrow}$.

The rest of the claims are verified by direct inspection. ∎

Informally, Theorem 7.1 can be interpreted as a statement that O-logic is essentially a syntactic variant of a relational (i.e., predicate-calculus-based) language. This suggests that the tug of war between relational and object-oriented languages does not have such deep theoretical underpinning as some people thought it does.

8. Skolemization, Herbrand Interpretations and Herbrand's Theorem

8.1. *Skolem Theorem*

Skolemization in O-logic is similar to its counterpart in the classical predicate calculus, since id-terms are essentially identical to terms in predicate calculus and quantification is defined similarly in both cases. For instance, $(\forall X\ \forall Y\ \exists Z)\ Y[attr_1 \rightarrow Z,\ attr_2 \rightarrow Y]$ can be Skolemized to $(\forall X\ \forall Y)\ Y[attr_1 \rightarrow f(X, Y),\ attr_2 \rightarrow Y]$, where f is a new binary function symbol. We therefore have an analogue of the Skolem theorem, whose proof is an easy adaptation from predicate calculus.

THEOREM 8.1 (cf. Skolem theorem). *Let ϕ be an O-formula and ϕ' be its Skolemization. Then ϕ is unsatisfiable if and only if so is ϕ'.*

As in predicate calculus, the Skolem theorem leads to the notion of a *clause*. First, we can transform any O-formula into its prenex normal form. Then, by Skolemizing existential variables we can eliminate all existential quantifiers. Since all the remaining quantifiers are universal, we can drop them altogether. Finally, by transforming the formula into its conjunctive normal form, we end up with a set of disjunctions of P-literals and O-literals that is unsatisfiable if and only if the original O-formula was unsatisfiable. Each such (implicitly universally quantified) disjunction is called a *clause*.

8.2. *Herbrand Interpretations*

As we have seen, O-terms may be embedded in other O-terms to an arbitrary depth. However, a special kind of P-terms and O-terms, called *flat*, is most common. A *flat* O-term is either a typing O-term, an is-a O-term, or a complex O-term of the form $T[..., funA_i \rightarrow T_i, ..., setA_j \twoheadrightarrow \{...S_{j,k}...\}, ...]$, where all the terms T_i, $S_{j,k}$ inside the brackets and braces are id-terms. A *flat* P-term is a P-term of the form $p(R_1, ..., R_n)$, where each R_i is an id-term. It is easy to see that every P-term and O-term is equivalent to a conjunction of flat terms. For instance,

$$john[works \rightarrow toys[mngr \rightarrow gene], child \twoheadrightarrow \{jack[age \rightarrow 3], mary[age \rightarrow 4]\}]$$

is equivalent to

$$john[works \rightarrow toys, child \twoheadrightarrow \{jack, mary\}] \wedge toys[mngr \rightarrow gene] \wedge jack[age \rightarrow 3] \wedge mary[age \rightarrow 4].$$

Restricting attention to flat O-terms can greatly simplify the notation used in the O-logic proof theory and, except for the examples, flat O-terms are assumed until the end of this paper.

Given an O-logic language **L** with a set of constructors **F**, its *Herbrand universe* is **F***—the set of ground id-terms. The *Herbrand base*, denoted *HB*(**L**), is the set of all ground O-logic molecules in the language **L** (including O-terms, equality, typing, is-a, and P-terms).

A *Herbrand interpretation* is a subset of the Herbrand base that is *closed* under the logical entailment "$\models$." The closedness requirement is a matter of convenience; it is adopted because ground molecules may imply other molecules in a nontrivial way, e.g., $\{a[funA \rightarrow b], a[funA \rightarrow c]\} \models (b = c)$ or $\{\mathbf{p} : \mathbf{q}, \mathbf{q} : \mathbf{r}\} \models \mathbf{p} : \mathbf{r}$. This is reminiscent of the situation in predicate calculus with equality, where the condition is that the equations in any Herbrand interpretation must form a congruence relation.

It is easy to verify that any Herbrand interpretation H for a language **L** satisfies the following *closure properties*:

(1) Equality atoms in H define a congruence relation:

Reflexivity. For all terms p in $\mathbf{F}^*$, $(p=p)\in H$.

Symmetry. If $(p=q)\in H$, then $(q=p)\in H$.

Transitivity. If $\{p=q,\ q=r\}\subseteq H$, then $(p=r)\in H$.

Substitutivity. If molecules $s=t$ and L are in H and L' is the result of replacing one occurrence of s in L by t, then L' is also in H. Here, L can be either an O-term or a P-term.

(2) For is-a and typing O-terms, the following holds:

Reflexivity. $\mathbf{p}:\mathbf{p}\in H$, for each $\mathbf{p}\in\Sigma$.

Transitivity. If $\{t:\mathbf{q},\ \mathbf{q}:\mathbf{r}\}\subseteq H$, then $t:\mathbf{r}\in H$, for any $t\in\mathbf{F}^*$ and $\mathbf{q}, \mathbf{r}\in\Sigma$. If $\{\mathbf{p}:\mathbf{q},\ \mathbf{q}:\mathbf{r}\}\subseteq H$, then $\mathbf{p}:\mathbf{r}\in H$, for any $\mathbf{p}, \mathbf{q}, \mathbf{r}\in\Sigma$.

(3) For every ground id-term t, $t[\ ]$ is in H.

(4) For complex O-terms. If $t[funA\rightarrow r_1]$, $t[funA\rightarrow r_2]\in H$, then $(r_1=r_2)\in H$.

(5) For every O-term t, t is in H if and only if every *constituent atom* of t is in H.

The constituent *atoms* mentioned in (5) are defined as follows:

- Every typing O-term, $T:\mathbf{p}$, or an is-a O-term, $\mathbf{q}:\mathbf{p}$, is its own constituent atom.
- For a flat complex O-term $R=T[..., funA_i\rightarrow V_i, ..., setA_j\twoheadrightarrow\{S_{j,1}, ..., S_{j,k}\}, ...]$, its constituent atoms are: $T[funA_i\rightarrow V_i]$, $T[setA_j\twoheadrightarrow\{\ \}]$, $T[setA_j\twoheadrightarrow\{S_{j,1}\}], ..., T[setA_j\twoheadrightarrow\{S_{j,k}\}]$, for each functional attribute $funA_i$ and each set-valued attribute $setA_j$ occurring in R.
- Every flat P-term is its own constituent atom.

LEMMA 8.1. *A molecule R is satisfied by a semantic structure I and a variable assignment μ if and only if so is every atom of R.*

Proof. Immediately follows from the definitions. ∎

Formula satisfaction in Herbrand interpretations is now defined as follows:

- For a ground molecule A, $H\models A$ (resp., $H\models\neg A$) if and only if $A\in H$ (resp., $A\notin H$).
- For a ground clause $C\equiv L_1\vee\cdots\vee L_n$, $H\models C$ if and only if $H\models L_i$ for some $1\leqslant i\leqslant n$.
- For a non-ground clause, $H\models C$ if and only if $H\models C'$ for all ground instances C' of C.

For a set of clauses **S**, if every clause in **S** is true with respect to H, then we say that H is a *Herbrand model* of **S**.

8.3. *Reduction to Herbrand Interpretations*

There is a natural correspondence between Herbrand interpretations and semantic structures. Given a semantic structure I, the Herbrand interpretation H_I *corresponding* to I is simply the set of all ground molecules that are true in I. Conversely, for a Herbrand interpretation H over the Herbrand universe $\mathbf{F}^*$, its corresponding semantic structure, $I_H = \langle U, I_{\mathbf{F}}, I_{\to}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi} \rangle$, is defined as follows:

(1) $U \stackrel{\text{def}}{=} \mathbf{F}^*/_{=}$; that is, U is the quotient of $\mathbf{F}^*$ induced by the equations in H. The equivalence class of t is denoted by $[t]$.

(2) $I_{\mathbf{F}}(t) = [t]$, for every 0-ary symbol $t \in \mathbf{F}$.

(3) $I_{\mathbf{F}}(f)([t_1], ..., [t_k]) = [f(t_1, ..., t_k)]$, for every k-ary $(k \geqslant 1)$ function symbol f in $\mathbf{F}$.

(4) $I_{\to}(funA)([t]) = [s]$, if $t[funA \to s] \in H$; undefined, otherwise.

(5) $I_{\twoheadrightarrow}(setA)([t]) = \{[s] \mid t[setA \twoheadrightarrow \{s\}] \in H\}$, if $t[setA \twoheadrightarrow \{\ \}] \in H$; undefined, otherwise.

(6) $I_{\Sigma}(\mathbf{p}) = \{[s] \mid s : \mathbf{p} \in H\}$.

(7) $I_{\Pi}(p) = \{\langle [t_1], ..., [t_n] \rangle \mid p(t_1, ..., t_n) \in H\}$, for each n-ary predicate $p \in \Pi$.

(8) $\leqslant_{\Sigma} = \{\langle \mathbf{p}, \mathbf{q} \rangle \mid \mathbf{p} : \mathbf{q} \in H\}$.

We remark that if in (5) $t[setA \twoheadrightarrow \{\ \}] \notin H$ then there is no s such that $t[setA \twoheadrightarrow \{s\}] \in H$ and so $I_{\twoheadrightarrow}(setA)([t])$ is undefined. In contrast, when $t[setA \twoheadrightarrow \{\ \}] \in H$ then $I_{\twoheadrightarrow}(setA)([t])$ is defined, but this set may still be empty if there is no s such that $t[setA \twoheadrightarrow \{s\}] \in H$.

It is easy to see that $I_H = \langle U, I_{\mathbf{F}}, I_{\to}, I_{\twoheadrightarrow}, I_{\Sigma}, \leqslant_{\Sigma}, I_{\Pi} \rangle$ defined above is indeed a semantic structure. The next proposition establishes a connection between general and Herbrand models in O-logic.

PROPOSITION 8.1. *Let* $\mathbf{S}$ *be a set of clauses. Then* $I \models \mathbf{S}$ *if and only if* $H_I \models \mathbf{S}$. *In particular,* $\mathbf{S}$ *is unsatisfiable if and only if* $\mathbf{S}$ *has no Herbrand model.*

Proof. "Only-if." It suffices to show that if $\mathbf{S}$ has a Herbrand model H, then the corresponding semantic structure, I_H, is a model of $\mathbf{S}$. First we claim that for an atom τ of any molecule, if $H \models \tau$ then $I_H \models \tau$. Suppose, for definiteness, that $\tau = t[funA \to s]$ (the cases of typing and is-a O-terms as well as those of set-valued attributes and P-terms are similar). Suppose $H \models \tau$. Then, by the definition of I_H, $I_{\to}(funA)([t]) = [s]$. Hence, $I_H \models \tau$.

By Lemma 8.1, a semantic structure satisfies a ground molecule if and only if it satisfies every atom of that molecule. It follows from the above that $H \models t$ implies $I_H \models t$ for ground molecules. This result extends to arbitrary clauses and sets of clauses by structural induction. Therefore, since $H \models \mathbf{S}$, we have $I_H \models \mathbf{S}$.

"If." Assume that $\mathbf{S}$ has a model I. We will show that its corresponding Herbrand

intrepretation, H_I, is a Herbrand model for S. By definition, H_I consists of ground molecules that are true in I. So, for any ground molecule α, if $I \models \alpha$ then $H_I \models \alpha$. Again, by structural induction, this result extends to arbitrary clauses and the sets thereof. ∎

8.4. *Herbrand's Theorem*

In the first-order predicate calculus, a set S of clauses is unsatisfiable if and only if so is some finite subset of ground instances of clauses in S. This statement is usually referred to as *Herbrand's theorem.*

It is easily seen from the construction in Theorem 7.1 that, given an O-logic Herbrand interpretation H, its predicate calculus counterpart, $\Phi(H)$, is a Herbrand interpretation for $\Phi(\mathbf{S})$. Furthermore, from the closure properties for Herbrand interpretations (Section 8.2) it follows that Φ is a surjective mapping from the set of O-logic Herbrand models of S onto the set of Herbrand models of $\Phi(\mathbf{S})$. The O-logic analogue of Herbrand's theorem now follows from these observations and from the Herbrand's theorem in predicate calculus.

THEOREM 8.2 (cf. Herbrand's theorem). *A clause set* S *is unsatisfiable if and only if so is some finite subset of ground instances of clauses in* S.

Herbrand's theorem is a basis for resolution-based sound and complete proof theories in predicate calculus [17]. The next section presents one such theory for O-logic.

9. PROOF THEORY

9.1. *Ordering and Merging of O-terms*

We can define a pre-order, $\leqslant$, on (possibly nonground) O-terms as follows. Let T_1 and T_2 be a pair of O-terms or P-terms. We say that T_1 is a *subterm* of T_2, denoted $T_1 \leqslant T_2$, if and only if every atom of T_1 is also an atom of T_2. For instance, $T[attr \twoheadrightarrow \{X\}] \leqslant T[attr \twoheadrightarrow \{X, Y\}]$, but $T[attr \twoheadrightarrow \{Z\}]$ is not a subterm of $T[attr \twoheadrightarrow \{X, Y\}]$.

In case of complex O-terms, $T_1 \leqslant T_2$ intuitively means that for the same object *obj*, T_1 asserts fewer properties of *obj* than T_2 does. In cases of is-a and typing O-terms or P-terms, $T_1 \leqslant T_2$ simply means that T_1 and T_2 are identical. The following lemma presents a semantic characterization of this ordering:

LEMMA 9.1. *Let T be a flat O-term or a P-term. Then for every semantic structure I and every variable assignment v, $I \models_v T$ if and only if for every subterm T' of T, $I \models_v T'$.*

Proof. Directly follows from the definition of "$\models$." ∎

Given a pair of complex O-terms T_1 and T_2 with the same id, their *merge* is a complex O-term S such that the set of all atoms of S equals the union of the sets of atoms of T_1 and T_2. Merge of a pair of terms may not be unique. For instance, $T[funA \to c,\ setA \twoheadrightarrow \{a\}]$ and $T[funA \to d,\ setA \twoheadrightarrow \{b\}]$ have two merges: $T[funA \to c,\ funA \to d,\ setA \twoheadrightarrow \{a\},\ setA \twoheadrightarrow \{b\}]$ and $T[funA \to c,\ funA \to d,\ setA \twoheadrightarrow \{a, b\}]$. However, we distinguish the *canonical merge*, which *is* unique.

A merge of T_1 and T_2 is *canonical*, denoted $merge(T_1, T_2)$, if and only if it has no repeated occurrences of the same set-valued attribute (repeated functional attributes are allowed). For the above example, the second of the merges is canonical, while the first one is not. It is easy to see that $merge(T_1, T_2)$ is unique up to a permutation of atoms and of id-terms in the ranges of set-valued attributes. The following is a consequence of Lemma 9.1.

COROLLARY 9.1. $I \models_v merge(T_1, T_2)$ if and only if $I \models_v T_1$ and $I \models_v T_2$.

9.2. *Unification*

For id-terms and flat P-terms, unification is defined in the same way as in predicate calculus: Let T_1 and T_2 be a pair of id-terms or P-terms. A substitution σ is called a *unifier* of T_1 and T_2, if and only if $\sigma(T_1) = \sigma(T_2)$.

A pair of typing O-terms, $T : \mathbf{p}$ and $S : \mathbf{q}$, *unifies* if the id-terms T and S unify and $\mathbf{p}$ is identical to $\mathbf{q}$. Is-a O-terms unify if they are identical.

Unification of complex O-terms is defined somewhat differently: Rather than requiring terms to be identical after applying the substitution, we merely ask that one term will be a *subterm* of the other. Let T_1 and T_2 be a pair of complex O-terms. A substitution σ is a *unifier of* T_1 *into* T_2 (note the asymmetry!) if and only if $\sigma(T_1) \leqslant \sigma(T_2)$.

A unifier σ of T_1 and T_2 (T_1 *into* T_2, in case of complex O-terms) is *most general* (abbreviated *mgu*) if for every other unifier μ of T_1 and/into T_2 such that $\sigma = \gamma \circ \mu$ for some substitution γ, there is a substitution δ such that $\mu = \delta \circ \sigma$. Here, $\gamma \circ \mu$ and $\delta \circ \sigma$ are functional compositions, e.g., $\gamma \circ \mu(t) = \gamma(\mu(t))$. This definition of mgu coincides with the classical one, except that it uses a different notion of unifier, defined above.

For id-terms, is-a or typing O-terms, and for P-terms, mgu is always unique up to an equivalence and the above definition is tantamount to the standard one. However, for complex O-terms there may be several (but a finite number of) mgu's. For example, there are two mgu's (namely, $\langle X/a \rangle$ and $\langle X/b \rangle$) of $a[setA \twoheadrightarrow \{X\}]$ into $a[setA \twoheadrightarrow \{a, b\}]$. Therefore, resolving away a pair of such O-terms may yield *several* different resolvents, which differs from the situation in predicate calculus. This multitude of resolvents is an inevitable consequence of the presence of set-valued attributes in O-logic. In related theories that deal with sets as first-class citizens, multiple mgu's are common when sets are to be unified (e.g., [13, 39]).

In O-logic, the role of the unique mgu of the classic predicate calculus is taken over by the notion of a *complete set* of mgu's. Given a pair of O-terms P and Q, we say that a set Ω of mgu's of P into Q is *complete*, if and only if for every

substitution θ, such that $\theta(P) \leqslant \theta(Q)$, there exists $\sigma \in \Omega$, such that $\theta = \gamma \circ \sigma$, for some substitution γ. It follows from the definition that, given a pair of O-terms, any two complete sets of mgu's are identical up to an equivalence.[5] To preserve continuity of the presentation, we postpone the description of an algorithm for finding a complete set of mgu's until Appendix A.

9.3 *Inference Rules*

The existence of *some* proof theory for O-logic follows from Theorem 7.1. However, translating O-terms into predicate calculus is not a good way to do business since most of the "computational hints" embedded in the structure of O-terms are lost in this translation. The proof theory presented next operates directly on O-terms and takes their structure into account.

Our deductive system for O-logic consists of the *reflexivity* axioms for equality and is-a O-terms and of several inference rules: *resolution*, *factoring*, *paramodulation*, *merging*, *functionality*, *elimination*, and *transitivity*. All these rules assume that the clauses are *standardized apart*, that is, variables are renamed so that when a rule is applied to a pair of clauses, the clauses do not share common variables. However, standardization does not preclude rules from being applied to pairs of instances of the same clause and even—in case of ground rules—to pairs of identical copies of the same clause. Also, to simplify the language, we will sometimes talk about unifying a P-term (resp., typing or is-a O-terms) *into* another P-term (resp., typing or is-a O-terms), even though unification in this case is symmetric.

REFLEXIVITY AXIOMS. (1) $(\forall X)\, X = X$.

(2) $\mathbf{p} : \mathbf{p}$, *for every* $\mathbf{p} \in \Sigma$.

The main workhorse in our deduction system is the resolution rule, which is an adaptation from predicate calculus. The resolution rule seeks to resolve a negative literal in one clause with a positive literal in another clause.

Resolution. Consider a pair of (not necessarily ground) O-logic clauses $C \stackrel{\text{def}}{=} \neg P \vee C_1$ and $C' \stackrel{\text{def}}{=} Q \vee C_2$ with no common variables and suppose that there is an mgu θ of P into Q. Then

$$\theta(C_1) \vee \theta(C_2) \tag{10}$$

is a *binary resolvent* of C and C'. Note that the resolution rule is asymmetric when the literals involved in the operation are complex O-terms.

It is well known from predicate calculus that binary resolution does not suffice, and one also needs factoring and paramodulation rules.

Factoring. Let $C \stackrel{\text{def}}{=} T \vee T' \vee C'$ (resp., $\neg T \vee \neg T' \vee C'$) be a clause, such that T is unifiable into T' with the mgu θ. Then $\theta(T \vee C')$ (resp., $\theta(\neg T' \vee C')$) is called

[5] Sets of unifiers Ω_1 and Ω_2 are equivalent if (1) for each $\sigma_1 \in \Omega_1$ there is $\sigma_2 \in \Omega_2$ and substitutions γ, δ such that $\sigma_1 = \gamma \circ \sigma_2$ and $\sigma_2 = \delta \circ \sigma_1$; and (2) for each $\sigma_2 \in \Omega_2$ there is $\sigma_1 \in \Omega_1$ with the properties symmetric to those described in (1).

a *factor* of C. Observe the difference between the positive and the negative cases of factoring: in the positive case, the "smaller" term T remains in the factor, while in the negative case it is the larger term, T', that survives.

Paramodulation. Let $C \stackrel{\text{def}}{=} L[T] \vee C_1$ and $C' \stackrel{\text{def}}{=} (R = S) \vee C_2$ be a pair of clauses with no common variables, where $L[T]$ is a literal containing an id-term T. If T and R are unifiable id-terms with the mgu θ, then

$$\theta(L[S]) \vee \theta(C_1) \vee \theta(C_2) \tag{11}$$

is a *paramodulant* of C and C'. Here, $L[S]$ denotes L with one occurrence of T replaced by S. Paramodulation applies to O-terms as well as P-terms.

Unlike predicate calculus, resolution, factoring, and paramodulation do not suffice for a complete proof procedure. Consider the following pair of O-terms: $a[funA \rightarrow b,\ setA \twoheadrightarrow \{c\}]$ and $a[funA \rightarrow d,\ setA \twoheadrightarrow \{e\}]$. Clearly, $a[funA \rightarrow b,\ funA \rightarrow d,\ setA \twoheadrightarrow \{c, e\}]$ is logically entailed by these terms, but this entailment cannot be proved using the rules introduced so far. In addition, the functional attribute *funA* in the above O-terms has two distinct values, b and d, for the same object a. Since functional attributes are interpreted as single-valued functions, it follows that $b = d$ must hold. To account for these situations, we introduce the *merging* and the *functionality* rules.

Merging. Let $C \stackrel{\text{def}}{=} P \vee C_1$ and $C' \stackrel{\text{def}}{=} Q \vee C_2$ be a pair of clauses with no variables in common, such that P and Q are positive complex O-terms. If the ids of P and Q are unifiable with the mgu θ, then the clause

$$merge(\theta(P), \theta(Q)) \vee \theta(C_1) \vee \theta(C_2) \tag{12}$$

is called a *merge* of C and C' (*merge*(..., ...) was introduced in Section 9.1).

Functionality. Let $C \stackrel{\text{def}}{=} T_1[..., funA \rightarrow R_1, ...] \vee C_1$ and $C' \stackrel{\text{def}}{=} T_2[..., funA \rightarrow R_2, ...] \vee C_2$ be a pair of clauses that share no variables. If T_1 and T_2 are unifiable with the mgu θ, then the *functional derivative* of C and C' is the clause

$$\theta(R_1 = R_2) \vee \theta(C_1) \vee \theta(C_2). \tag{13}$$

Still, all of the above inference rules do not form a complete proof theory. For instance, all degenerated O-terms of the form $T[\ ]$ (no attributes specified) are O-logic tautologies and yet they cannot be refuted using the rules given so far. Such O-terms are taken care of by the following *elimination* rule:

Elimination. Let $\neg T[\ ] \vee C$ be a clause. Its *eliminant* is the clause C. Note here that if C is empty, the elimination rule derives an empty clause $\square$, that is, $\square$ is the eliminant of the clause $\neg T[\ ]$.

To complete the picture, we need a rule that accounts for the transitivity of the is-a relationship.

Transitivity. Let $T:\mathbf{q}$ be a typing O-term and $\mathbf{q}:\mathbf{r}$ be an is-a O-term, respectively. Their *composition* is the typing O-term $T:\mathbf{r}$. For a pair of is-a O-terms $\mathbf{p}:\mathbf{q}$ and $\mathbf{q}:\mathbf{r}$, their *composition* is $\mathbf{p}:\mathbf{r}$. The *transitivity* rule is a rule that sanctions derivation of such compositions. More precisely, the transitivity rule derives $\mathbf{p}:\mathbf{r} \vee C_1 \vee C_2$ from $\mathbf{p}:\mathbf{q} \vee C_1$ and $\mathbf{q}:\mathbf{r} \vee C_2$; from $T:\mathbf{q} \vee C_1$ and $\mathbf{q}:\mathbf{r} \vee C_2$ it derives $T:\mathbf{r} \vee C_1 \vee C_2$.

Given a set $\mathbf{S}$ of clauses, a *deduction* of a clause C from $\mathbf{S}$, denoted $\mathbf{S} \vdash C$, is a finite sequence of clauses $D_1, ..., D_n$ such that $D_n = C$ and, for $1 \leqslant k \leqslant n$,

- $D_k \in \mathbf{S}$, or
- D_k is a reflexivity axiom, or
- D_k is derived from $D_1, ..., D_{k-1}$ via one of the above derivation rules.

A *refutation* of $\mathbf{S}$ is a deduction of the empty clause, $\square$, from $\mathbf{S}$.

9.4. *Adjustments for Inconsistency-Tolerant Attributes*

Inconsistency-tolerant attributes can be incorporated into the proof theory in two ways. The simplest one is to add the axioms (9) of Section 5 for every such attribute. The first of these axioms corresponds to the *reduction* rule in [31] and does not introduce new problems. The second axiom in (9) may result in a very inefficient computation, though. Instead of using this latter axiom, a better way would be to slightly change the definition of "unification into" in Section 9.2, which in turn will have implications for the resolution and factoring rules (since these are the only rules where this notion is used). However, no new inference rule will be needed.

In fact, we are only going to change the definition of the notion of a constituent *atom* of Section 8.2 so that the set of constituent atoms of the term $T[..., attr \rightsquigarrow \top, ...]$ will contain (among the rest) all the O-terms of the form $T[attr \rightsquigarrow S]$, for every id-term S.

This change in the definition affects the notion of a subterm, since now $T[attr \rightsquigarrow S] \leqslant T[attr \rightsquigarrow \top]$ (as every atom of the left-hand O-term is also an atom of the right-hand O-term). The implication for the resolution is that now $T[attr \rightsquigarrow \top] \vee C$ can be resolved with $\neg T[attr \rightsquigarrow S] \vee C'$, yielding $C \vee C'$. For the factoring rule, the new definition implies that $T[attr \rightsquigarrow S] \vee T[attr \rightsquigarrow \top] \vee C$ can be factored into $T[attr \rightsquigarrow S] \vee C$ and $\neg T[attr \rightsquigarrow S] \vee \neg T[attr \rightsquigarrow \top] \vee C$ into $\neg T[attr \rightsquigarrow \top] \vee C$.

9.5. *Soundness of the Proof Theory*

THEOREM 9.1 (Soundness of O-logic deduction). *If* $\mathbf{S} \vdash C$ *then* $\mathbf{S} \models C$.

Proof. It suffices to show that each of the aforementioned axioms and derivation rules is sound. As an example, we will establish soundness of the merging rule. Let M be a model for $P \vee C_1$ and $Q \vee C_2$. We need to show that $merge(\theta(P), \theta(Q)) \vee \theta(C_1) \vee \theta(C_2)$ is true in M, where θ is a mgu of the ids of P and Q.

If either $\theta(C_1)$ or $\theta(C_2)$ is true in M, then we are done. Otherwise, $M \models \theta(P)$ and $M \models \theta(Q)$. Since these O-terms have the same id, their merge exists. By Corollary 9.1, $M \models merge(\theta(P), \theta(Q))$. ∎

EXAMPLE 9.1. Consider again the database of Example 4.7:

(a) **workstudy** : **student**

(b) **workstudy** : **employee**

(c) *john* : **workstudy**

(d) $X[duty \twoheadrightarrow \{homework\}] \Leftarrow X$: **student**

(e) $Y[duty \twoheadrightarrow \{officework\}] \Leftarrow Y$: **employee**.

Let the query be

(f) $?-john[duty \twoheadrightarrow \{X\}]$.

Refutation of (f) can be done as follows. By transitivity of is-a O-terms, we derive

(g) *john* : **student**

(h) *john* : **employee**

from (a), (b), and (c). Resolving (g) with (d) and (h) with (e) yields

(i) $john[duty \twoheadrightarrow \{homework\}]$

(j) $john[duty \twoheadrightarrow \{officework\}]$.

Note that the O-term (i) and (j) have the same id, and hence can be merged:

(k) $john[duty \twoheadrightarrow \{homework, officework\}]$.

Finally, resolving (k) with the query (f) yields the empty clause. The complete set of mgu's here has just two elements: $\langle X/homework \rangle$ and $\langle X/officework \rangle$. Thus, the answer to (f) is $john[duty \twoheadrightarrow \{homework, officework\}]$.

9.6. *Completeness of the Ground Deduction*

Before proving completeness, we need the following auxiliary lemma:

LEMMA 9.2. *Let* **S** *be a set of ground O-logic literals. If* **S** *is unsatisfiable then there are molecules P and Q, such that* $P \leqslant Q$, $\neg P \in \mathbf{S}$, *and* $\mathbf{S} \vdash Q$.

Proof. Suppose to the contrary, that there are no molecules P and Q, such that $P \leqslant Q$, $\neg P \in \mathbf{S}$, and $\mathbf{S} \vdash Q$. We will show that then **S** is satisfiable.

Consider a subset of the Herbrand base: $D(\mathbf{S}) \stackrel{\text{def}}{=} \{P \in \mathbf{HB(L)} \mid$, *where P is a subterm of a molecule deducible from* **S**$\}$. We claim that $D(\mathbf{S})$ is closed under the logical entailment "$\models$" and thus is a Herbrand interpretation.

In Section 8.3, we have shown that for any Herbrand interpretation H there is a semantic structure I_H such that $H \models \mathbf{S}$ if and only if $I_H \models \mathbf{S}$. Applying the same

construction to $D(\mathbf{S})$ we get a semantic structure $M_{D(\mathbf{S})}$. This construction goes through and $M_{D(\mathbf{S})}$ is indeed a semantic structure because $D(\mathbf{S})$ is closed under "$\vdash$" and hence possesses the closure properties of Section 8.2. To show that $D(\mathbf{S})$ is also closed under "$\models$," we will prove that for any molecule P,

$$M_{D(\mathbf{S})} \models P \qquad \textit{if and only if} \quad P \in D(\mathbf{S}). \tag{14}$$

Soundness of the derivation rules provides for the "if" direction. For the "only-if" direction, consider the following cases:

(1) *P is a P-term or an is-a/typing O-term.* Depending on the type of P, cases (7), (8), or (6) in the construction of Section 8.3 can be used to show (14).

(2) *P is a complex O-term composed of the atoms $\alpha_1, ..., \alpha_n$.* Clearly, $M_{D(\mathbf{S})} \models P$ if and only if $M_{D(\mathbf{S})} \models \alpha_i$, $i = 1, ..., n$. Again, using cases (4) or (5) of Section 8.3 (depending on whether the attribute in α_j is functional or set-valued) we can show that $M_{D(\mathbf{S})} \models \alpha_j$ if and only if $\alpha_j \in D(\mathbf{S})$. Therefore, by the definition of $D(\mathbf{S})$, there are terms $Q_1, ..., Q_n$ deducible from $\mathbf{S}$ such that each α_j is a subterm of Q_j. Therefore, every atom of P is also an atom of $Q = merge(Q_1, ..., Q_n)$ and thus P is a subterm of Q. But Q is deducible from $D(\mathbf{S})$ since so is each of the Q_i. Hence, by the definition of $D(\mathbf{S})$, it follows that $P \in D(\mathbf{S})$, which proves (14).

Now, if $P \in \mathbf{S}$ is a positive literal then $P \in D(\mathbf{S})$; hence $D(\mathbf{S}) \models P$. By the assumption at the beginning of the proof, for every negative literal $\neg P$ in $\mathbf{S}$, P is not a subterm of any molecule in $D(\mathbf{S})$. So, $D(\mathbf{S}) \models \neg P$ (as $D(\mathbf{S})$ is a Herbrand interpretation). Thus, $D(\mathbf{S})$ satisfies every literal in $\mathbf{S}$, that is, $D(\mathbf{S})$ is a model for $\mathbf{S}$. ∎

THEOREM 9.2 (Completeness of the ground deduction). *If a set of ground clauses $\mathbf{S}$ is unsatisfiable then $\mathbf{S} \vdash \square$, where $\square$ denotes the empty clause.*

Proof. By Herbrand's theorem (Theorem 8.2), if $\mathbf{S}$ is unsatisfiable, then there exists a finite subset $\mathbf{T}$ of $\mathbf{S}$, such that $\mathbf{T}$ is unsatisfiable. We will show that $\mathbf{T} \vdash \square$ using a technique from [6]. The proof is carried out by induction on the number of excess literals in $\mathbf{T}$, denoted *excess*($\mathbf{T}$), where

$$\textit{excess}(\mathbf{T}) \stackrel{\text{def}}{=} (\textit{the number of occurrences of literals in } \mathbf{T}) - (\textit{the number of clauses in } \mathbf{T}).$$

Basis. $\textit{excess}(\mathbf{T}) = 0$. Then the number of clauses in $\mathbf{T}$ equals the number of literals in $\mathbf{T}$. Therefore, every clause in $\mathbf{T}$ consists of either a positive or a negative literal. Since $\mathbf{T}$ is unsatisfiable, by Lemma 9.2, there is a negative literal $\neg P$ in $\mathbf{T}$, and a positive literal Q deducible from $\mathbf{T}$, such that $P \leqslant Q$. Then, by first deriving Q from $\mathbf{T}$ and resolving $\neg P$ with Q, we can obtain the empty clause.

Induction step. $\textit{excess}(\mathbf{T}) = n > 0$. In this case, there must be a clause C in $\mathbf{T}$ that contains more than one literal. Let $\mathbf{T} = \{C\} \cup T'$ and $C = L \vee C'$ (C' is not $\square$ since we have assumed that C contains more than one literal). By the distributivity law,

$\{L \vee C'\} \cup \mathbf{T}'$ is unsatisfiable if and only if $\mathbf{T}_1 = \{L\} \cup \mathbf{T}'$ and $\mathbf{T}_2 = \{C'\} \cup \mathbf{T}'$ are unsatisfiable. Since $excess(\mathbf{T}_1) < n$ and $excess(\mathbf{T}_2) < n$, by the inductive hypothesis, $\mathbf{T}_1 \vdash \square$ and $\mathbf{T}_2 \vdash \square$. Observe that if $\mathbf{T}_2 \vdash \square$, then by applying to $\mathbf{T}$ the deduction sequence that produces $\square$ from $\mathbf{T}_2$, we can derive either L or $\square$. If $\square$ is so produced, then we are done. Otherwise, L is produced. Using the same deduction sequence that derived $\square$ from $\mathbf{T}_1$, we can derive $\square$ from $\mathbf{T} \cup \{L\}$. Combining the two deductions yields a refutation of $\mathbf{T}$. ▮

9.7. *Lifting the Ground Deduction*

To prove completeness of non-ground deduction, we first establish an O-logic analogue of the lifting lemma [17]. It is shown in Appendix A that for every pair of nonground O-terms, there is a finite complete set of mgu's. Alas, the size of this set may be exponential, which is inevitable in the presence of sets. However, the next proposition and the comments in Section 9.8 show that the complexity of set-unification is polynomial in some important special cases, such as logic programs.

PROPOSITION 9.1. *Let P, Q be possibly nonground O-terms with the same* id. *There is an algorithm that finds a complete set, Ω, of mgu's of P into Q. Ω is finite, but in the worst case it may be exponential in the size of P and Q. However*:

(a) *Ω has only one element, if P, Q are P-terms or typing/is-a O-terms, or if Q is a complex O-term without repeated occurrences of the same attribute and with only singleton sets*;

(b) *The size of Ω is polynomial in the size of Q, if P is a complex O-term whose number of set-valued attributes and the size of its sets is bounded by a global constant that is independent of the size of Q.*

Proof. See Lemma A1 and the discussion in Appendix A. ▮

LEMMA 9.3 (Lifting lemma). *Suppose A and B are clauses with no variables in common, and let A', B' be some instances of A and B, respectively. If C' is derived from A' and B' using one of the inference rules of Section* 9.3, *then applying the same rule to some factors*[6] *$\bar{A}$ and $\bar{B}$ of A and B, respectively, yields a clause $\bar{C}$ such that C' is an instance of $\bar{C}$.*

Proof. The proofs for all inference rules are pretty much similar; we will only consider a slightly more involved case of the resolution rule.

Suppose A' and B' can be resolved with a mgu μ. Without loss of generality, we assume that there is a negative literal $\neg P'$ in A' and a positive literal Q' in B', such that $\mu(P') \leqslant \mu(Q')$. We also assume that A' and B' (resp., A and B) do not share variables. Since A' and B' are instances of A and B (and variables are not shared), there is a substitution θ, such that $A' = \theta(A)$ and $B' = \theta(B)$. Let $\neg P_1, ..., \neg P_k$ be all the literals in A that are mapped by θ into $\neg P'$, and let $Q_1, ..., Q_l$ be all the

[6] As in predicate calculus, this is the only place where the factoring rule comes into picture.

literals in B that are mapped into Q'. Let $\bar{A}$ (resp., $\bar{B}$) be a factor of A (resp., B) that splices the P_i's (resp., the Q_i's) together into the term $\bar{P}$ (resp., $\bar{Q}$). Thus, there is a substitution η such that $P' = \theta(P_1) = \eta(\bar{P})$ and $Q' = \theta(Q_1) = \eta(\bar{Q})$. Since $\mu \circ \theta(P_1) = \mu(P') \leqslant \mu(Q') = \mu \circ \theta(Q_1)$, it follows that

$$\mu \circ \eta(\bar{P}) = \mu(P') \leqslant \mu(Q') = \mu \circ \eta(\bar{Q}).$$

Therefore, $\mu \circ \eta$ is a unifier of $\bar{P}$ into $\bar{Q}$. Let Ω be a complete set of unifiers of $\bar{P}$ into $\bar{Q}$. By the definition of complete sets of mgu's, there is a mgu $\sigma \in \Omega$ such that $\mu \circ \eta = \gamma \circ \sigma$, for some γ. Thus, using σ as the mgu, we can resolve $\bar{A}$ and $\bar{B}$ to obtain $\bar{C}$. It is straightforward to verify that C' is an instance of $\bar{C}$. ▮

THEOREM 9.3 (Completeness of O-logic deduction). *If a set* S *of clauses is unsatisfiable, then there is a refutation from* S.

Proof. It is easy to see that S is unsatisfiable if and only if the set of all its ground instances S′ is unsatisfiable. By completeness of ground deduction (Theorem 9.2), there is a refutation from S′. Then, with the help of the lifting lemma, we can lift this ground refutation to a nonground refutation from S. ▮

9.8. *Remarks on the Proof Theory*

It is known that one of the major obstacles to an efficient implementation of logic formalisms that deal with sets is the complexity of set-unification. There is a major difference between set-unification in O-logic and that in LDL, COL, and ELPS [2, 13, 39]. For a pair of sets of id-terms S and S', unification in O-logic finds the most general substitution σ such that $\sigma(S) \subseteq \sigma(S')$. This is known as the "first-order subsumption problem" [24] and is NP-hard. In comparison, the corresponding unification problem for LDL, COL, and ELPS requires strict equality: $\sigma(S) = \sigma(S')$. Although both problems have equally bad worst-case complexity, the subsumption problem seems easier, at least conceptually. In the rest of this subsection we will argue that O-logic has computational advantages in the important case of logic programs.

The proposed solution to set-unification in LDL is to rewrite programs at compile-time, replacing set-unification by an exponential number of rules that are free of set-terms [49].

In O-logic, we have two options. Since there are simple algorithms for set-subsumption, we can use them directly. For instance, it is generally accepted that the size of each individual rule in any logic program **P** is small compared to the size of facts and the number of rules in **P**. Therefore, we can assume that sizes of O-terms in rule-bodies are globally bounded by a small constant. Since body-literals are unified *into* the head-literals (head-literals include the fact-literals in the database), it follows from the discussion in Appendix A that the complexity of unification in O-logic is polynomial in the maximum of sizes of the head literals in **P**. Measuring complexity in this manner is akin to Vardi's *data complexity* for queries [56].

Another way to do unification is to follow the lead of LDL and rewrite the rules at compile time. For instance, we can simplify literals in rule-bodies by splitting them into atoms. Clearly, this does not increase the number of clauses, while their sizes may increase only by a constant factor. Since unification of an atom A into O-term B is polynomial in the size of the O-term B (see Appendix A), performing every single resolution step in the transformed program is relatively inexpensive. A potentially expensive operation is factoring, since it involves unification of pairs of positive O-terms and the latter cannot be split into atoms without an exponential increase in the program size. Fortunately, factoring is not needed to evaluate logic programs, as in the classical case.

We thus see that in O-logic it is possible to make unification a polynomial operation by blowing up the program size by a constant factor, which stands in sharp contrast with LDL, where the program size may be blown up exponentially.[7]

The last remark in this subsection concerns the equality predicate. In classic logic programming, programs do not contain equality in rule-heads, and the so-called *freeness axioms* for equality [47] are assumed. Under the freenes axioms in O-logic, the paramodulation rule becomes redundant. The functionality rule can now be applied in conjunction with the merging rule in such a way that whenever a functional attribute is found to be multiply-defined, the empty clause will be derived. For instance, merging the terms $T[..., funA \rightarrow r_1, ...]$ and $T[..., funA \rightarrow r_2, ...]$ in this way would yield inconsistency, provided that r_1 and r_2 are ground and non-unifiable. Likewise, for inconsistency-tolerant attributes, $T[attr \rightsquigarrow r_1]$ and $T[attr \rightsquigarrow r_2]$ would yield $T[attr \rightsquigarrow \top]$ here.

10. A Logic Programming Semantics

The semantics of O-logic (that is, the relation "$\models$") described so far is *monotonic*, which means that $\mathbf{S} \models \phi$ implies $\mathbf{S} \cup \mathbf{S}' \models \phi$, where $\mathbf{S}$ and $\mathbf{S}'$ are sets of O-formulas and ϕ is an O-formula. However, in databases and logic programming it is customary to supply only the true data, assuming (roughly) that a fact is false if its truth cannot be established. The reason for not specifying negative data explicitly is, among other things, that the size of such data is very large and often is even infinite. The semantics corresponding to this latter kind of reasoning is called *nonmonotonic*. In the classic case, the nonmonotonic semantics of logic programs is built on top of the ordinary monotonic semantics of predicate calculus by restricting the logical implication to certain subclasses of the class of all models, called *intended* or *canonic* models. Shoham [50] showed that this principle underlies a large number of known nonmonotonic theories.

In classic logic programming, a number of different semantics have been proposed (e.g., [3, 25, 37, 44, 55]). Most of these semantics agree with each other

[7] It should be clear, however, that even though the size of O-logic programs does not undergo significant increase, the number of unification steps may well be exponential, as in LDL.

on the class of *locally stratified* programs. The semantics of such programs is based on so called *perfect models* and is described in [44]. In this section we show how this semantics can be adapted to O-logic.

10.1. *Minimal Models*

For a starter, we consider Horn-clause programs and define the notion of minimal models and the renowned $T_{\mathbf{P}}$ operator.

Let **L** be an O-logic language and I, I' be a pair of interpretations. We write $I \ll^{\min} I'$ if whenever $I \models \psi$ for some molecule ψ then also $I' \models \psi$. For Herbrand interpretations, it follows from the definitions that $I \ll^{\min} I'$ if and only if $I \subseteq I'$. Given a set **S** of clauses, a model I is *minimal* if and only if for any other model I' of **S**, $I' \ll^{\min} I$ implies $I \ll^{\min} I'$.

In general, a minimal model may not be unique, even in the Herbrand case. However, as in the classic case, when **S** contains only definite Horn clauses then there is a unique Herbrand model. The proof can be obtained either directly (along the lines of [42]) or by twiddling with Theorem 7.1. For Horn clauses, the minimal Herbrand model is considered to be the canonic one and the program semantics is defined relatively to this unique model. More precisely, the nonmonotonic entailment relation "$\approx$" is introduced such that $\mathbf{S} \approx \phi$ if and only if $M \models \phi$, where M is the unique minimal model of **S**.

A well-known result about minimal models of Horn programs is that they coincide with the least fixed point of the one-step-derivation operator $T_{\mathbf{P}}$. Similar result holds for O-logic: Let **P** be a set of (O-logic) Horn clauses and H be a Herbrand interpretation. Then, $T_{\mathbf{P}}(H)$ is the smallest Herbrand interpretation that contains the following set of molecules:

$$\{p \mid p \Leftarrow body \text{ is a ground instance of a clause in } \mathbf{P} \text{ such that } H \models body\}.$$

Such a smallest interpretation always exists; the reader can easily verify that intersection of any number of Herbrand interpretations is closed under "$\models$" and thus, again, is a Herbrand interpretation.

Iterations of $T_{\mathbf{P}}$ are now defined as follows. Let $\perp$ be the smallest Herbrand interpretation for **L**, that is, $\perp$ contains only tautological O-terms, such as $\mathbf{p} : \mathbf{p}$, $a = a$, and $t[\,]$. Then

$$T_{\mathbf{P}}^0 = \perp;$$

$$T_{\mathbf{P}}^{n+1} = T_{\mathbf{P}}(T_{\mathbf{P}}^n);$$

$$T_{\mathbf{P}}^{\omega} = \bigcup_{i=1}^{\infty} T_{\mathbf{P}}^i.$$

It is left to the reader to verify (either directly or with the use of Theorem 7.1) that $T_{\mathbf{P}}^{\omega}$ coincides with the unique minimal Herbrand model of **P**.

We also mention that the standard query optimization techniques developed for deductive databases (e.g., [12, 21, 30, 57]) can be applied to O-logic, again, not

surprisingly in view of Theorem 7.1. There are new twists to this problem, however, because O-logic programs contain more semantic information (such as functional attributes) than their classic counterparts. We will not discuss query optimization any further in this paper.

10.2. *Perfect Models*

When clauses have negation in the body then the minimal-model semantics is no longer adequate. An appropriate semantics based on stratification was first proposed in [7, 54] and then generalized by Przymusinski in [44], who called it the *perfect-model* semantics. We now show how these ideas can be applied in the context of O-logic. The general principle (here and in other cases, such as stable-model and well-founded semantics) is that whenever predicates are used in the classic case, attributes should be used in O-logic. For instance,

$$X[trusts \twoheadrightarrow \{Y\}] \Leftarrow \neg X[suspects \twoheadrightarrow \{Y\}]$$

is stratified in O-logic, even though the object X recursively depends on itself through negation. On the other hand,

$$X[beats \twoheadrightarrow \{Y\}] \Leftarrow plays(X, Y) \,\&\, \neg Y[beats \twoheadrightarrow \{X\}]$$

is not stratified because of the negative recursion through the attribute *beats*.

There is a difficulty, however, related to the use of equality. In [44], the perfect-model semantics was introduced under the assumption that equality is axiomatized by the *freeness axioms*, which say that two terms are equal if and only if they are identical. Without this assumption, the standard definitions may give counter-intuitive results. For instance, a direct and simple-minded application of the definitions in [44] to the program

$$a = b \Leftarrow \neg p(b)$$
$$p(a)$$

would yield a unique perfect model $\{p(a), p(b), a = b\}$. The problem here is that $p(b)$ is not supported by any rule and, in fact, the equation $a = b$ (and thus also $p(b)$) was derived based on the assumption that $p(b)$ is false! In the classic theory of logic programs, this anomaly is avoided by banning the equality predicate from rule-heads. However, this does not work well for O-logic because, as noted in Section 5, equality can be generated via functional attributes. Besides, we believe that being able to equate symbols is important for knowledge representation. Fortunately, the solution to this problem is simple: It suffices to preclude the equality atoms from being dependent negatively on other literals.

Formally, given an O-logic language **L** and a program **P**, a *dependency graph* $D_{\mathbf{L}}(\mathbf{P})$ is defined as follows: Let $\mathbf{P}^*$ be the set of all ground instances of the rules in **P**. The *nodes* of $D_{\mathbf{L}}(\mathbf{P})$ correspond to the ground atoms constructible in **L**. A *possitive* arc $\phi \xrightarrow{+} \psi$ is in $D_{\mathbf{L}}(\mathbf{P})$ whenever $\mathbf{P}^*$ has a rule $\bar{\psi} \Leftarrow \ldots \,\&\, \bar{\phi} \,\&\, \ldots$, where ϕ

and ψ are constituent atoms of the molecules $\bar{\phi}$ and $\bar{\psi}$, respectively. A *negative* arc $\phi \xrightarrow{-} \psi$ is in $D_L(\mathbf{P})$ whenever $\mathbf{P}^*$ has a rule $\bar{\psi} \Leftarrow \ldots \& \neg\bar{\phi} \& \ldots$, Note that both ϕ and ψ can be equations. In addition to the arcs listed above, $D_L(\mathbf{P})$ has the following *positive* arcs: whenever some complex atomic O-terms $a[attr \to b]$ and $a[attr \to c]$ appear in the rule-heads in $\mathbf{P}^*$, we draw the arcs $a[attr \to b] \xrightarrow{+} (b = c)$ and $a[attr \to c] \xrightarrow{+} (b = c)$.

A directed path or a cycle in $D_L(\mathbf{P})$ is negative if it contains at least one negative arc. A program $\mathbf{P}$ is *locally stratified* if:

(1) $D_L(\mathbf{P})$ has no negative path leading into an equation-node; and

(2) $D_L(\mathbf{P})$ has no negative cycles.

A *stratification* is a function

- *level*: {atoms of $\mathbf{P}^*$} $\to \mathbf{N}$

such that if $D_L(\mathbf{P})$ has a directed path from ϕ to ψ then $level(\phi) \leqslant level(\psi)$. If there is a negative such path then $level(\phi) < level(\psi)$.

The *preference* order on semantic (not necessarily Herbrand) structures is now introduced as follows: M is *preferable* to N, denoted $M \ll^{\text{perf}} N$, if whenever there is an atom ϕ such that $M \models \phi$ and $N \not\models \phi$ then there is an atom ψ such that $level(\psi) < level(\phi)$, $N \models \psi$, and $M \not\models \psi$. Models that are minimal with respect to "$\ll^{\text{perf}}$" are called *perfect*. The following result can be proved either directly (along the lines of [44]) or with the help of Theorem 7.1:

PROPOSITION 10.1. *Every locally stratified O-logic program has a unique Herbrand perfect model.* ∎

10.3. *Equality-Minimal Models*

In Section 5 we saw that functional attributes can force equality among id-terms. For instance,

$$\begin{aligned} &sally[father \to bob] \\ &sally[father \to chair(cs)] \end{aligned} \tag{15}$$

implies $bob = chair(cs)$. However, it well may be that $bob = chair(cs)$ is not an intended result but rather is a consequence of a programmer's error. Under the current semantics, the programmer would have to say $bob \neq chair(cs)$ explicitly to let the system known that a run-time error must be generated if both of the O-terms in (15) are derived. Similarly, if *father* were an inconsistency-tolerant attribute, we would like to enable the system to derive $sally[father \rightsquigarrow \top]$ from

$$\begin{aligned} &sally[father \rightsquigarrow bob] \\ &sally[father \rightsquigarrow chair(cs)]. \end{aligned} \tag{16}$$

Again, at present, the inequality $bob \neq chair(cs)$ must be given explicitly to sanction such a derivation.

As mentioned in Section 5, one solution is to impose the closed world assumption on the equality predicate. To pull this plan through, we need to separate the equality definition from the rest of the program.

Formally, an *equality definition* is a set of Horn clauses that involve equations only. A *data definition* is any set of O-logic rules (possibly with negation in the body) that do not have equations in the rule-heads (but may have them in the body). A logic program $\mathbf{P} = EQ \cup DATA$ now consists of an equality definition EQ and a data definition $DATA$.

For a pair of semantic structures, we write $M \ll^{eq} N$, if for every ground equality-atom $a = b$, $M \models (a = b)$ implies $N \models (a = b)$.

A model M of $\mathbf{P}$ is *$\ll^{eq}$-minimal* if for every semantic structure N such that $N \models EQ$ and $N \ll^{eq} M$, it is the case that $M \ll^{eq} N$. Note that to be $\ll^{eq}$-minimal a model must possess a minimality property not just with respect to the models of $\mathbf{P}$ but also with respect to the models of EQ, which may be a much larger class of semantic structures.

The desired semantics is now obtained as follows. Let $\mathbf{P} = EQ \cup DATA$ be a locally stratified program. A model of $\mathbf{P}$ is *strongly perfect* if it is $\ll^{perf}$-minimal within the class of all $\ll^{eq}$-minimal models of $\mathbf{P}$.

Now, the program (15) above has no strongly perfect model, unless the programmer has explicitly stated that $bob = chair(cs)$. Indeed, in (15), the equality definition, EQ, is empty. Therefore, for every pair of nonidentical id-terms, a and b, $a \neq b$ holds in every $\ll^{eq}$-minimal semantic structure of (15); in particular, $bob \neq chair(cs)$ holds true. Therefore, no such structure can be a model of (15) and thus no strongly perfect model exists.

In contrast, (16) has a strongly perfect model in which $sally[father \rightsquigarrow \top]$ and therefore $sally[father \rightsquigarrow a]$ holds for every ground id term a (constructed in the language of the program in question). The reasoning here is similar to that for (15), but the conclusions are different: Every $\ll^{eq}$-minimal semantic structure for (16) satisfies $bob \neq chair(cs)$. The O-term $sally[father \rightsquigarrow \top]$ (hence $sally[father \rightsquigarrow a]$, for all a) can therefore be derived from axioms (9) in Section 5.

PROPOSITION 10.2. *Let $\mathbf{P} = EQ \cup DATA$ be a locally stratified program such that $DATA$ contains no ordinary functional attributes (but it may contain inconsistency-tolerant functional attributes). Then $\mathbf{P}$ has a strongly perfect model.*

Proof. Since $DATA$ has no functional attributes, $\mathbf{P}$ has $\ll^{eq}$-minimal models. In fact, it is easy to see that every ordinary perfect model of $\mathbf{P} \cup \{$axioms (9) of Section 5$\}$ is also a strongly perfect model of $\mathbf{P}$. ∎

11. CONCLUSION

We presented a logic that incorporates a number of most salient object-oriented features such as object identity, complex objects, and class/subclass classification.

We corrected the earlier problems with Maier's O-logic [43] and extended it in several directions. In particular, we incorporated sets and deduction and provided a sound and complete proof procedure. We also outlined a theory of logic programs in O-logic and demonstrated that the main ideas carry over from the classical case.

APPENDIX A

We now present an algorithm for finding a complete set of mgu's for a pair of O-terms. We remind (see Section 9.2) that it suffices to find just one complete set of mgu's, since any two such sets are equivalent to each other. It should be clear that since the structure of id-terms is exactly the same as that of the first-order terms, an mgu for any pair of unifiable id-terms can be found using any standard unification procedure. We will use this observation in Algorithm 1 below.

Let T be a complex O-term of the form $T'[funA_1 \to P_1, ..., funA_m \to P_m, setA_1 \twoheadrightarrow \{R_{1,1}, ..., R_{1,k_1}\}, ..., setA_n \twoheadrightarrow \{R_{n,1}, ..., R_{n,k_n}\}]$. We will use $attr(T)$ to denote the set $\{funA_1, ..., funA_m, setA_1, ..., setA_n\}$ of attributes listed in T; $atoms(T)$ will denote the set of constituent atoms of T. Also, if ψ is an atom of the form $T[attr \to P]$ or $T[attr \twoheadrightarrow \{P\}]$, then $val(\psi)$ is defined to be the singleton set $\{P\}$. If ψ is of the form $T[attr \twoheadrightarrow \{\ \}]$ then $val(\psi)$ is empty.

ALGORITHM 1. Finding a complete set of mgu's.

Input. Pair of O-terms T_1 and T_2 of the same kind (is-a, complex, typing).

Output. A complete set Ω of mgu's of T_1 into T_2.

Notation. Let Λ be a collection of all mappings $\{\lambda: atoms(T_1) \to atoms(T_2)\}$, where for every $\psi \in atoms(T_1)$ and $\lambda \in \Lambda$, $attr(\psi) = attr(\lambda(\psi))$.

Step 1

Put $\Omega := \{\ \}$.

If T_1 and T_2 are both is-a O-terms **then**

  If $T_1 = T_2$ **then** return $\Omega := \{id\}$,

    where id is a trivial mgu: $id(X) = X$ for all variables.

  else terminate with failure.

If $T_1 \stackrel{def}{=} S_1 : \mathbf{c}_1$ and $T_2 \stackrel{def}{=} S_2 : \mathbf{c}_2$ are typing O-terms **then**

  If $\mathbf{c}_1 \neq \mathbf{c}_2$ or S_1 and S_2 are nonunifiable **then** terminate with failure.

  else return $\Omega := \{mgu(S_1, S_2)\}$. % the standard mgu

Step 2

Let S_1 and S_2 be the ids of T_1 and T_2.

If S_1 and S_2 are unifiable **then** set $\theta := mgu(S_1, S_2)$. % the standard mgu

else terminate with failure.

Step 3
If $attr(T_1) \not\subseteq attr(T_2)$ **then** terminate with failure.
For each mapping $\lambda \in \Lambda$ **do**
begin
Put $\sigma_\lambda := \theta$.
For each atom ψ in $atoms(T_1)$ **do**
Let ψ' be $\lambda(\psi)$.
If $val(\psi) = \varnothing$ **then** do nothing.
else if $val(\psi') = \varnothing$ **then** terminate with failure.
else
Let $val(\psi)$ be $\{P\}$ and $val(\psi')$ be $\{Q\}$.
If $\sigma_\lambda(P)$ and $\sigma_\lambda(Q)$ are unifiable
then put $\sigma_\lambda := mgu(\sigma_\lambda(P), \sigma_\lambda(Q)) \circ \sigma_\lambda$. % standard unification
else discard this σ_λ and
jump out of the inner **for each**-loop to select another λ
Set $\Omega := \Omega \cup \{\sigma_\lambda\}$
end

Step 4
Return Ω, a complete set of mgu's.

The complexity of Algorithm 1 is exponential, since the number of mappings in the loop in Step 3 may be as large as $\prod_{satA \in sattr(T_1)} n_2(setA)^{n_1(setA)}$, where $n_i(setA)$ is the number of id-terms in the set pointed to by the attribute $setA$ in T_i; $sattr(T_1)$ denotes the collection of all set-valued attributes appearing in T_1. It is easy to see, however, that when every attribute in T_2 has only one occurrence and every set-valued attribute in T_2 is associated with either the empty set or a singleton set (i.e., when $n_2(setA) \leqslant 1$), there is only one mapping in Λ and therefore only one mgu. The same is true if T_1 has no attributes at all (i.e., when $n_1(setA) = 0$), since then Algorithm 1 exits in Step 2. Furthermore, if the number of elements in $sattr(T_1)$ is bounbed by a global constant (independent of T_2) and if for all $setA$ in $sattr(T_1)$ the number $n_1(setA)$ is also bounded, then there is only a polynomial number of choices (in the size of T_2) in the outer loop in Step 3 of Algorithm 1. Hence, the size of Ω is also polynomial. Section 9.8 explains that this latter case is characteristic of logic programming.

EXAMPLE A1. Let $T_1 = X[setA \twoheadrightarrow \{Y\}]$ and $T_2 = a[setA \twoheadrightarrow \{b, c\}]$ be given as an input to Algorithm 1. Then the output will be a pair of mgu's $\theta_1 = \langle X/a, Y/b \rangle$ and $\theta_2 = \langle X/a, Y/c \rangle$. The reader can verify that $\{\theta_1, \theta_2\}$ is a complete set of mgu's of T_1 into T_2.

LEMMA A1. *Algorithm* 1 *finds a complete set of mgu's of* T_1 *into* T_2.

Proof. Clearly, all elements of Ω are mgu's of T_1 into T_2, so we will only show that Ω is complete. Let θ be a substitution such that $\theta(T_1) \leqslant \theta(T_2)$. By the

definition of the ordering on O-terms (see Section 9.1), there is a mapping λ: $atoms(T_1) \to atoms(T_2)$, such that for every $\psi \in atoms(T_1)$, the attributes of ψ and $\lambda(\psi)$ are identical and $\theta(val(\psi)) \subseteq \theta(val(\lambda(\psi)))$. Hence, $\lambda \in \Lambda$. The substitution σ_λ constructed in the outer **for each**-loop in Step 3 is a most general substitution such that $val(\psi) \subseteq val(\lambda(\psi))$ for each $\psi \in atoms(T_1)$. So, θ must be less general than σ_λ, i.e., $\theta = \gamma \circ \sigma_\lambda$ for some substitution γ.

Summarizing, we have shown that (1) each element of Ω is a mgu; and (2) if there is a unifier θ of T_1 into T_2, then θ is an instance of some element in Ω. Hence, Ω is a complete set of mgu's of T_1 into T_2. ∎

ACKNOWLEDGMENTS

This paper would have never materialized without the insightful work of David Maier [43]. The discussions with Weidong Chen, Georg Lausen, and David Warren helped to shape up the framework. Catriel Beeri provided us with insights into the nature of sets in LDL, which helped clarify the difference between the treatment of sets in LDL and O-logic. Thanks to Paris Kanellakis and Jeff Ullman for their valuable comments on this paper. The following people have contributed time to discuss ideas presented herein: Chyouhwa Chen, Sanjay Manchanda, Ester Shilcrat, T. Krishnaprasad, and Jiyang Xu. We are also grateful to the anonymous referees for a number of suggestions.

REFERENCES

1. S. ABITEBOUL AND S. GRUMBACH, COL: A logic-based language for complex objects, *in* "Workshop on Database Programming Languages, Roscoff, France, Sept. 1987," pp. 253–276.
2. S. ABITEBOUL AND C. BEERI, "On the Power of Languages for Manipulation of Complex Objects," Rapport de Recherche INRIA, 1988; *ACM Trans. Database Systems*, to appear.
3. S. ABITEBOUL AND V. VIANU, Procedural and declarative database update languages, *in* "Proceedings, ACM SIGACT–SIGMOD Symp. on Principles of Database Systems, 1988," pp. 240–250.
4. S. ABITEBOUL AND P. C. KANELLAKIS, Object identity as a query language primitive, *in* "Proceedings, ACM–SIGMOD Intl. Conf. on Management of Data, 1989," pp. 159–173.
5. H. AIT-KACI AND R. NASR, LOGIN: A logic programming language with built-in inheritance, *J. Logic Programming* 3, No. 1 (1986), 185–215.
6. R. ANDERSON AND W. W. BLEDSOE, A linear format resolution with merging and a new technique for establishing completeness, *J. Assoc. Comput. Mach.* 17, No. 3 (1970), 525–534.
7. K. R. APT, H. BLAIR, AND A. WALKER, Towards a theory of declarative knowledge, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 89–148, Morgan Kaufmann, Los Altos, CA, 1988.
8. F. BANCILHON, A logic-programming/object-oriented cocktail, *in* "SIGMOD Record," Sept. 1986.
9. F. BANCILHON AND S. N. KHOSHAFIAN, A calculus of complex objects, *in* "Proceedings, ACM SIGACT–SIGMOD Symp. on Principles of Database Systems, 1986," pp. 53–59.
10. F. BANCILHON, Object-oriented database systems, *in* "Proceedings, ACM SIGACT–SIGMOD Symp. on Principles of Database Systems, 1988," pp. 152–162.
11. J. BANERJEE, W. KIM, AND K. C. KIM, Queries in object-oriented databases, *in* "Proceedings, 4th Intl. Conf. on Data Engineering, Los Angeles, CA, February 1988."
12. C. BEERI AND R. RAMAKRISHNAN, On the power of magic, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1987," pp. 269–283.

13. C. BEERI, S. NAQVI, O. SHMUELI, AND S. TSUR, "Sets and Negation in a Logic Database Language (LDL)," MCC Report, 1987.
14. C. BEERI, R. NASR, AND S. TSUR, "Embedding Ψ-Terms in a Horn-Clause Logic Language," MCC Tech. Rep. ACA-ST-050-88, January 1988.
15. C. BEERI, Data models and languages for databases, *in* "2nd Int. Conf. on Database Theory (ICDT), Bruges, Belgium, 1988," pp. 19–40, Lect. Notes in Comput. Sci., Vol. 326, Springer-Verlag, New York/Berlin, 1988.
16. C. BEERI, Formal models for object-oriented databases, *in* "Proceedings, 1st Intl. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan, Dec. 1989," pp. 370–395.
17. C. L. CHANG AND R. C. T. LEE, "Symbolic Logic and Mechanical Theorem Proving," Academic Press, New York/London, 1973.
18. W. CHEN, M. KIFER, AND D. S. WARREN, Hilog as a platform for database languages (or why predicate calculus is not enough), *in* "Database Programming Languages" (R. Hull, R. Morrison, and D. Stemple, Eds.), Morgan Kaufmann, Los Altos, CA, 1989.
19. W. CHEN AND D. S. WARREN, C-logic for complex objects, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1989," pp. 369–378.
20. W. CHEN, M. KIFER, AND D. S. WARREN, Hilog: A first-order semantics for higher-order logic programming constructs, *in* "Proceedings, North American Conference on Logic Programming, Cleveland, Ohio, Oct. 1989."
21. S. DIETRICH-WAGNER AND D. S. WARREN, Extension tables: Memo relations in logic programming, *in* "Symposium on Logic Programming, San Francisco, CA, Sept. 1987," pp. 264–273.
22. H. B. ENDERTON, "A Mathematical Introduction to Logic," Academic Press, New York/London, 1972.
23. R. FIKES AND T. KEHLER, The role of frame-based representation in reasoning, *Comm. ACM* **28** (1985), 904–920.
24. M. R. GAREY AND D. S. JOHNSON, *in* "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1978.
25. M. GELFOND AND V. LIFSCHITZ, The stable model semantics for logic programming, *in* "Fifth Conference and Symposium on Logic Programming, 1988," pp. 1070–1080.
26. M. GYSSENS AND D. VAN GUCHT, The powerset operator as an algebraic tool for understanding least fixpoint semantics in the context of nested relations, *in* "Proceedings, ACM-SIGMOD Intl. Conf. on Management of Data, 1988."
27. P. J. HAYES, The logic of frames, *in* "Frame Conception and Text Understanding" (D. Metzing, Ed.), pp. 46–61, de Gruyter, Berlin, 1979.
28. R. HULL, A survey of theoretical research on typed complex database objects, *in* "Databases" (J. Paradaens, Ed.), pp. 193–256, Academic Press, New York, 1987.
29. S. N. KHOSHAFIAN AND G. P. COPELAND, Object identity, *in* "OOPSLA-86, 1986," pp. 406–416.
30. M. KIFER AND E. L. LOZINSKII, A framework for an efficient implementation of deductive database systems, *in* "Proceedings, 6th Advanced Database Symposium, Tokyo, Japan, Aug. 1986," pp. 103–106.
31. M. KIFER AND J. WU, A logic for object-oriented logic programming (Maier's O-logic revisited), *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, March 1989," pp. 379–393.
32. M. KIFER AND G. LAUSEN, F-logic: A higher-order language for reasoning about objects, inheritance, and schema, *in* "Proceedings, ACM-SIGMOD Intl. Conf. on Management of Data, June 1989," pp. 134–146.
33. M. KIFER AND E. L. LOZINSKII, RI: A logic for reasoning with inconsistency, *in* "Symposium on Logic in Computer Science (LICS), June 1989," pp. 253–262.
34. M. KIFER, G. LAUSEN, AND J. WU, "Logical Foundations of Object-Oriented and Frame-Based Languages," Tech. Rep. 90/14, Department of Computer Science, SUNY at Stony Brook, June 1990, to appear in *Journal of ACM*.
35. M. KIFER AND E. L. LOZINSKII, A logic for reasoning with inconsistency, *J. Autom. Reasoning* **9** (1992), 179–215.

36. W. KIM, J. BANERJEE, H. CHOU, J. F. GARZA, AND D. WOELK, Composite object support in an object-oriented database system, *in* "Proceedings, OOPSLA-87, 1987."
37. P. G. KOLAITIS AND C. H. PAPADIMITRIOU, Why not negation by fixpoint, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1988," pp. 231–239.
38. G. M. KUPER AND M. Y. VARDI, A new approach to database logic, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1984."
39. G. M. KUPER, "An extension of LPS to Arbitrary Sets," IBM research report, 1987.
40. C. LECLUSE AND P. RICHARD, Modeling inheritance and genericity in object-oriented databases, *in* "2nd Int. Conf. on Database Theory (ICDT), Bruges, Belgium, 1988," pp. 223–238, Lect. Notes in Comput. Sci., Vol. 326, Springer-Verlag, New York/Berlin, 1988.
41. C. LECLUSE, P. RICHARD, AND F. VELEZ, O_2, an object-oriented data model, *in* "Proceedings, ACM-SIGMOD Intl. Conf. on Management of Data, 1988," pp. 424–433.
42. J. W. LLOYD, "Foundations of Logic Programming, 2nd ed., Springer-Verlag, New York/Berlin, 1987.
43. D. MAIER, A logic for objects, *in* "Proceedings, Workshop on Foundations of Deductive Databases and Logic Programming, Washington, Aug. 1986," pp. 6–26.
44. T. C. PRZYMUSINSKI, On the declarative semantics of deductive databases and logic programs, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 193–216, Morgan Kaufmann, Los Altos, CA, 1988.
45. M. A. ROTH, H. F. KORTH, AND A. SILBERSCHATZ, "Extended Algebra and Calculus for $\neg$1NF Relational Databases," Tech. Rep. 84–36, Dept. of Computer Science, University of Texas at Austin, 1985.
46. H. RUBINSKI, On first-order databases, *ACM Trans. Database Systems* **12**, No. 3 (1987), 325–349.
47. J. C. SHEPHERDSON, Negation in logic programming, *in* "Foundations of Deductive Databases and Logic Programming" (J. Minker, Ed.), pp. 19–88, Morgan Kaufmann, Los Altos, CA, 1988.
48. O. SHMUELI, Decidability and expressiveness aspects of logic queries, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1987," pp. 237–249.
49. O. SHMUELI, S. TSUR, AND C. ZANIOLO, Rewriting of rules containing set terms in a logical data language (LDL), *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1988," pp. 15–28.
50. Y. SHOHAM, A semantic approach to nonmonotonic logics, *in* "Reading in Nonmonotonic Reasoning" (M. L. Ginsberg, Ed.), pp. 227–249, Morgan Kaufmann, 1987.
51. M. STEFIK AND D. G. BOBROW, Object-oriented programming: Themes and variations, *The AI Mag.*, Jan. (1986), 40–62.
52. J. D. ULLMAN, Database theory: Past and future, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1987," pp. 1–10.
53. J. D. ULLMAN, "Principles of Database and Knowledge-Base Systems," Computer Sci., Rockville, MD, 1988.
54. A. VAN GELDER, Negation as failure using tight derivations for general logic programs, *in* "Symposium on Logic Programming, 1986."
55. A. VAN GELDER, K. ROSS, AND J. S. SCHLIPF, Unfounded sets and well-founded semantics for general logic programs, *in* "Proceedings, ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, Austin, TX, 1988," pp. 221–231.
56. M. Y. VARDI, The complexity of relational query languages, *in* "Proceedings, 14th ACM Symposium on Theory of Computation, May 1982," pp. 137–146.
57. L. VIEILLE, Recursive axioms in deductive databases: The query–subquery approach, *in* "Proceedings, 1st Conf. on Expert Database Systems, Charleston, SC, 1986," pp. 179–196.
58. P. WEGNER, The object-oriented classification paradigm, *in* "Research Directions in Object-Oriented Programming" (B. Sriver and P. Wegner, Eds.), pp. 479–560, MIT Press, Cambridge, MA, 1987.
59. C. ZANIOLO, H. AIT-KACI, D. BEECH, S. CAMMARATA, L. KERSCHBERG, AND D. MAIER, "Object Oriented Database Systems and Knowledge Systems," MCC Tech. Rep. DB–038–85, MCC, 1985.